ELSEVIER

Theoretical Computer Science 241 (2000) 83–114

Theoretical Computer Science

www.elsevier.com/locate/tcs

Learning bounded-width OBDDs

Atsuyoshi Nakamura

Theory NEC Laboratory, RWCP[1]*, c/o C & C Media Research Laboratories, NEC Corporation, 4-1-1 Miyazaki Miyamae-ku, Kawasaki 216, Japan*

Abstract

In this paper, we study exact learnability of bounded-width ordered binary decision diagrams (OBDDs) when no ordering of the variables is given and learning is conducted by way of equivalence queries and membership queries. We present a learning algorithm for width-2 OBDDs, an algorithm which uses $O(n^3)$ equivalence queries alone, where n is the number of variables. We also present a learning algorithm for width-2 OBDDs that uses $O(n)$ *proper* equivalence queries and $O(n^2)$ membership queries. Further, we show a negative result: that there are no polynomial-time algorithms capable of learning width-3 OBDDs from proper equivalence queries alone.

1. Introduction

Ordered binary decision diagrams (OBDDs), which are also known as (ordered) branching programs, are a useful representation of Boolean functions. This type of representation has been the subject of recent study in a wide variety of fields [5]. In the field of computational learning theory, there have already been studies on query learnability of μ-branching programs[2] [9], query learnability of OBDDs with a given variable ordering [8], and PAC-learnability of bounded-width branching programs [7, 6].[3]

In this paper, we focus on exact learnability of *bounded-width* OBDDs. This has previously been studied by Ergün et al. [7] who have shown that width-2 branching programs are PAC-learnable and that PAC-learning of width-3 branching programs is at least as hard as PAC-learning of DNF. We note that they assumed in their study that the ordering of variables in a target OBDD is *given*. Here we assume that the ordering is *unknown*. This assumption makes the learning difficult because with respect to a function representable by a width-2 OBDD with one ordering, there will

E-mail address: atsu@sbl.cl.nec.co.jp (A. Nakamura).

[1] RWCP: Real World Computing Partnership.

[2] Branching programs in which no variable appears more than once.

[3] The branching programs considered in [6] are not restricted to ordered ones.

0304-3975/00/$ - see front matter
PII: S0304-3975(99)00267-4

be a different ordering for which this function may not necessarily be representable by an OBDD whose width is a polynomial of the number of variables. (see Appendix a). The difficulty of learning OBDDs with an unknown best ordering has been shown by Gavaldà and Guijarro [8], and it has seemed more feasible to us to limit our investigation to bounded-width OBDDs with an unknown best ordering.

As a learning framework, we use Angluin's query learning model [1], which allows two kinds of queries: equivalence queries and membership queries. We present a learning algorithm for width-2 OBDDs[4] (with an unknown ordering) using $O(n^3)$ equivalence queries alone, where n is the number of variables. Our algorithm, which represents a slight modification of Simon's learning algorithm [10] for decision trees, asks equivalence queries with a hypothesis decision list belonging to a certain function class. This class is a proper superset of the width-2 OBDD class. Thus the equivalence queries asked by the algorithm are *not* proper.

While exact learnability of the width-2 OBDD class by way of *proper* equivalence queries alone has yet to be shown, we can do so when membership queries are also allowed to be asked, and in a later section, we present a learning algorithm for the width-2 OBDD class using $O(n)$ proper equivalence queries and $O(n^2)$ membership queries. Our algorithm uses a *layer list* as a hypothesis. A layer list is a new representation by which every function representable by width-2 OBDDs can be represented uniquely.

We also show a negative result concerning proper learnability of width-3 OBDDs: by modifying the proof for the read-once formulas of Angluin et al. [2], we are able to show that there are no polynomial-time algorithms capable of learning width-3 OBDDs from proper equivalence queries alone.

The outline of the paper is as follows. In Section 2, we provide definitions and notations used in this paper. In Section 3, we explain the two representations that our algorithms use: one is a layer list and the other is a decision list. In Sections 4 and 5, we present our query learning algorithms for width-2 OBDDs. In Section 6, we describe our hardness result for width-3 OBDDs. Finally, in Section 7, we conclude this paper with some open questions.

2. Preliminaries

Let $x_1,\ldots,x_n$ be Boolean variables and $X=\{(x_1,\ldots,x_n):x_1,\ldots,x_n\in\{0,1\}\}$. A *binary decision diagram* (BDD) is a directed acyclic graph with one root (source) node and two sink nodes: a 0-labelled sink node and a 1-labelled sink node. Each internal node is labelled with a Boolean variable and has two outgoing edges, one labelled 0 and the other 1. In an *ordered BDD* (OBDD), there must exist a linear ordering '$<$' of variables such that $x_i<x_j$ if there is an edge from a node labelled x_i to a node labelled x_j. An OBDD can be used as a representation of a Boolean function

[4] In a recent independent study, Bergadano et al. have shown a more general result [4] (see Section 4).

on X calculated as follows: for an assignment $x_1 = a_1, \ldots, x_n = a_n$, start from its root node, select the a_i-labelled outgoing edge at a node labelled x_i, and regard the label of the sink node reached finally as the value of the function. For an OBDD D and an assignment e, Path(D, e) denotes the subgraph of D whose nodes and edges are passed in this process.

We refer to the set of nodes labelled by a variable x as the *x-level*. Let D be an OBDD with an ordering '$<$', $\{x_{i_1}, \ldots, x_{i_k}\}$ be the set of variables labelling the nodes in D, and assume $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$. We say that D is a *levelled* OBDD if every outgoing edge from the x_{i_j}-level enters the $x_{i_{j+1}}$-level for any $j \in \{1, \ldots, k-1\}$ and every outgoing edge from the x_{i_k}-level enters one of the sink nodes. The *width* of a levelled OBDD D is defined as the maximum number of nodes contained at any level in D. A *width-k* OBDD is a levelled OBDD having width of at most k.

A *decision list* is a list L of pairs $[(f_0, g_0), \ldots, (f_d, g_d)]$, where all f_i and g_i are Boolean functions on X and f_d is a 1-valued constant function $\mathbf{1}$. We say that d is the *length* of L. The decision list L represents a Boolean function h_L on X calculated as follows: $h_L(\boldsymbol{x})$ for $\boldsymbol{x} \in X$ is $g_j(\boldsymbol{x})$, where (f_j, g_j) is the leftmost item in L satisfying $f_j(\boldsymbol{x}) = 1$. For classes $\mathscr{F}$ and $\mathscr{G}$ of Boolean functions on X with $\mathbf{1} \in \mathscr{F}$, $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$ denotes the class of decision lists of the form $[(f_0, g_0), \ldots, (f_d, g_d)]$ such that $f_i \in \mathscr{F}$ and $g_i \in \mathscr{G}$ for all $i \in \{0, \ldots, d\}$. We define the *alternation level* of the jth item in L as follows. The jth item (f_j, g_j) is at level 0 if $g_i = g_j$ on $\{\boldsymbol{x} \in X : f_i(\boldsymbol{x}) = f_j(\boldsymbol{x}) = 1\}$ for all $i \in \{0, \ldots, j-1\}$. Let $X_i = \{\boldsymbol{x} \in X : \exists j[f_j(\boldsymbol{x}) = 1, (f_j, g_j)$ is at level $i]\}$. Assume that the j_kth item is the last item in L at level $k \geqslant 0$. Then, the jth item (f_j, g_j) for $j > j_k$ is at level $k+1$ if $g_i = g_j$ on $\{\boldsymbol{x} \in X - \bigcup_{h=0}^{k} X_h : f_i(\boldsymbol{x}) = f_j(\boldsymbol{x}) = 1\}$ for all $i \in \{j_k+1, \ldots, j-1\}$. We also say that the alternation level of f_j in $[(f_0, g_0), \ldots, (f_d, g_d)]$ is i when (f_j, g_j) is at level i. Note that any swapping of items at the same alternation level leaves h_L unchanged. We say that L has l *alternations* if the last alternation level of L is level l. Throughout the paper, we assume that the last alternation level contains (f_d, g_d) only. Any decision list with $f_d = \mathbf{1}$ can be converted to such a form by removing unnecessary items. The alternation level of f which does not appear in L is defined as the last one. This is because adding (f, g_d) to L as the last item does not change the function represented by L.

As previously noted, we use as our learning framework the *query learning model* proposed by Angluin [1]. In this model, the goal of a learning algorithm is to identify a target function f using *equivalence queries* and *membership queries*. An *equivalence query* asks if the unknown target function f is equivalent to a hypothesis h; if so the answer 'YES' is returned, otherwise a counterexample $e(f(e) \neq h(e))$ is returned. An equivalence query is *proper* if a hypothesis h used in the query is also a member of the function class from which a target function f is chosen. A *membership query* asks the value of $f(a)$ of a target function f for an assignment a.

In the sequel, $\overline{1}$ denotes 0 and $\overline{0}$ denotes 1. For variable x, $\bar{x}$ denotes the negation of x. We define x^0 as $\bar{x}$ and x^1 as x. For assignment e, $e_{\bar{x}}$ denotes the assignment obtained from e by flipping the bit assigned to x. We let $x(e)$ denote the value assigned to x by assignment e.

3. Representations

3.1. Layer list representation

In this subsection, we define a new representation for width-2 OBDDs, called a *layer list*, by which every function representable by width-2 OBDDs is represented uniquely. The *layer list* of a width-2 OBDD can be obtained by applying the following two-step conversion to the OBDD:

Step 1: Conversion of a width-2 OBDD into one in *normal form*, and

Step 2: Conversion of a *normal-form* width-2 OBDD into a *layer list*.

3.1.1. Conversion into a normal-form width-2 OBDD

Let D be an arbitrary width-2 OBDD. Every node in D can be placed on a pair of parallel lines such that (1) either line contains at most one sink node and at most one node in every level, and (2) the nodes appear on each line according to the variable ordering of D. We call the line containing 0-labelled sink node the *0-lane* and the other line the *1-lane*. (See Fig. 1.)

In this paper, OBDDs are drawn so as to place their 0-lane on the left side. The appearance of D depends on which node is placed on the 0-lane at each level. Thus, we introduce a *normal form* of width-2 OBDDs, in which the lane each node should be placed on is uniquely determined.

Assume that one of the two lanes is assigned to every node of D. By changing its lane assignments without changing the function it represents, we convert D into a normal-form width-2 OBDD. We use two procedures, *Body-Conversion* and *Root-Conversion* for this task. In procedure Body-Conversion, the lane assignments to nodes in each level are changed and unnecessary levels are removed. After procedure Body-Conversion, the lane assignments except for that to the root node do not depend on its

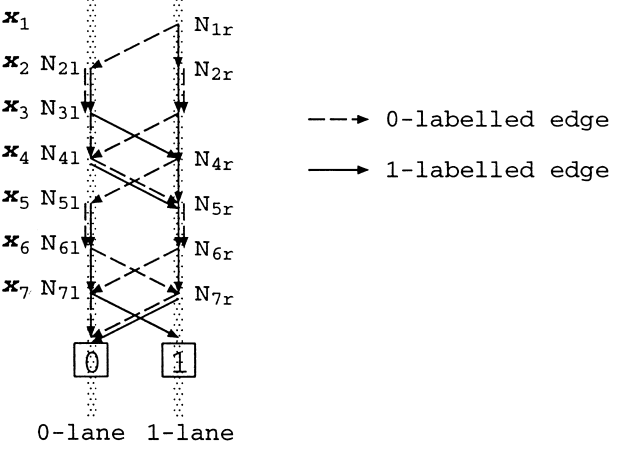


Fig. 1. Example of a width-2 OBDD (before normalization).

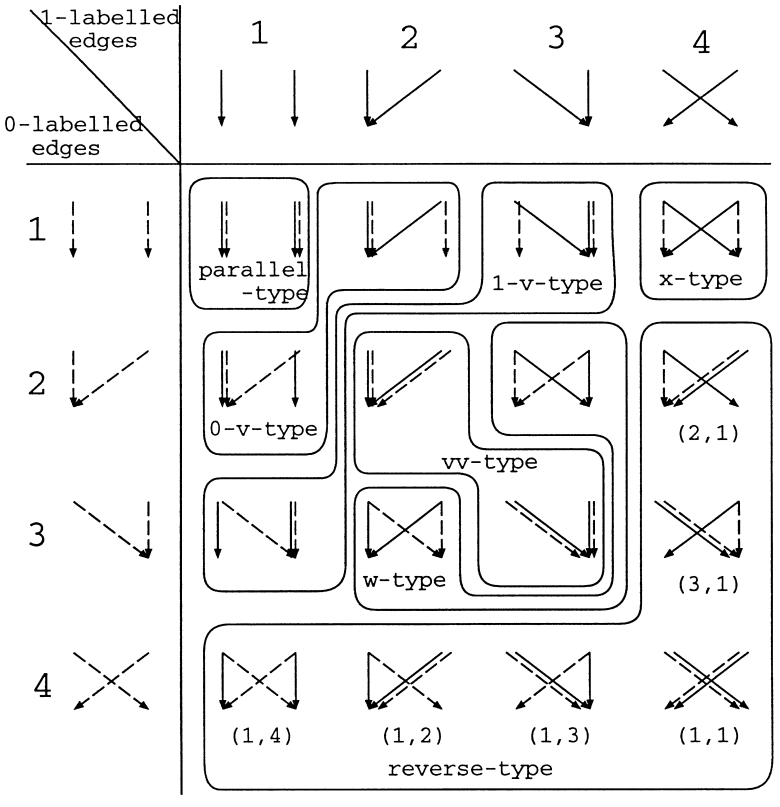
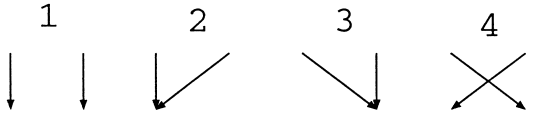


Fig. 2. Level types.

initial assignments. Procedure Root-Conversion uniquely settles the lane assignment to the root.

We first describe procedure Body-Conversion. First, note that, for $a = 0$ or 1, a pair of a-labelled edges outgoing from each level having two nodes belongs to one of four types below [7].



Thus, if we consider all edges outgoing from one level, there are 16 possible patterns. We classify these patterns and hence the levels into 7 types. (See Fig. 2.)

In procedure Body-Conversion, levels are processed one by one from below. Let L be the level which is processed currently. If L is a w-type, the procedure combines the two nodes into one. If L contains only one node, it removes all levels above L and stops. Thus, the procedure has no chance to process vv-type levels. If L is a reverse-type, all the lane assignments to the nodes in the levels above L and L itself

0. $i = -1$
1. $i = i + 1$
2. If L_i is w-type, combine the two nodes into one.
3. If L_i contains only one node, remove all levels above L_i and stop.
4. If L_i is reverse-type, reverse all the lane assignments to the nodes in the levels above L_i and L_i itself.
5. If L_i is parallel-type, remove L_i.
6. Go to 1.

Fig. 3. Procedure Body-Conversion.

are reversed. By reversing the lane assignments to the nodes in L, L is converted into one of four types, a parallel-type, a 0–v-type, a 1–v-type or an x-type. In Fig. 2, the coordinates shown in reverse-type entries indicate which entries they are converted into. If L is a parallel-type, the procedure removes that level. The precise description of the procedure is shown in Fig. 3. In Fig. 3, L_i denotes the ith level of D from below for $i \geqslant 1$, and L_0 denotes the level of sink nodes.

Procedure Body-Conversion converts the width-2 OBDD in Fig. 1 into the last one in Fig. 4. Note that, in the initial OBDD, the x_7-level, the x_6-level and the x_4-level are reverse-types, the x_5-level and the x_2-level are parallel-types, and the x_3-level is a w-type. The converted OBDD is composed of three types of levels, a 0–v-type, a 1–v-type and an x-type.

Procedure Root-Conversion decides which lane the root node should belong to. In order to do so, it is enough to decide whether the level of the root node is a 0–v-type, a 1–v-type or an x-type. For $a = 0$ or 1, the root node is on the $\overline{a}$-lane when the level is a-v-type. Note that $\overline{0} = 1$ and $\overline{1} = 0$ by definition. If the level is an x-type, the root node is to be on the same lane as the node pointed by the 0-labelled edge outgoing from the root node. Let L_l be the level of the root node. We decide the type of L_l as follows: L_l is the same type as the previous level L_{l-1} if $l \geqslant 2$, and L_l is an x-type if $l = 1$, namely, if the OBDD has only root node except the sink nodes. The detailed description of procedure Root-Conversion is given in Fig. 5.

Some conversion examples by procedure Root-Conversion are shown in Fig. 6.

We say that width-2 OBDD D is in *normal form* when the applications of procedure Body-Conversion and Root-Conversion do not change D. It follows from the definitions of the two procedures that all OBDDs are in normal form after the applications of the two procedures.

Proposition 1. *Let D' be the width-2 OBDD made from a width-2 OBDD D by applying Body-Conversion and Root-Conversion to D. Then, D' is in normal form.*

3.1.2. Conversion into a layer list

Are there different normal-form width-2 OBDDs representing the same function? The answer is yes. For example, the two OBDDs in Fig. 7 represent the same function.

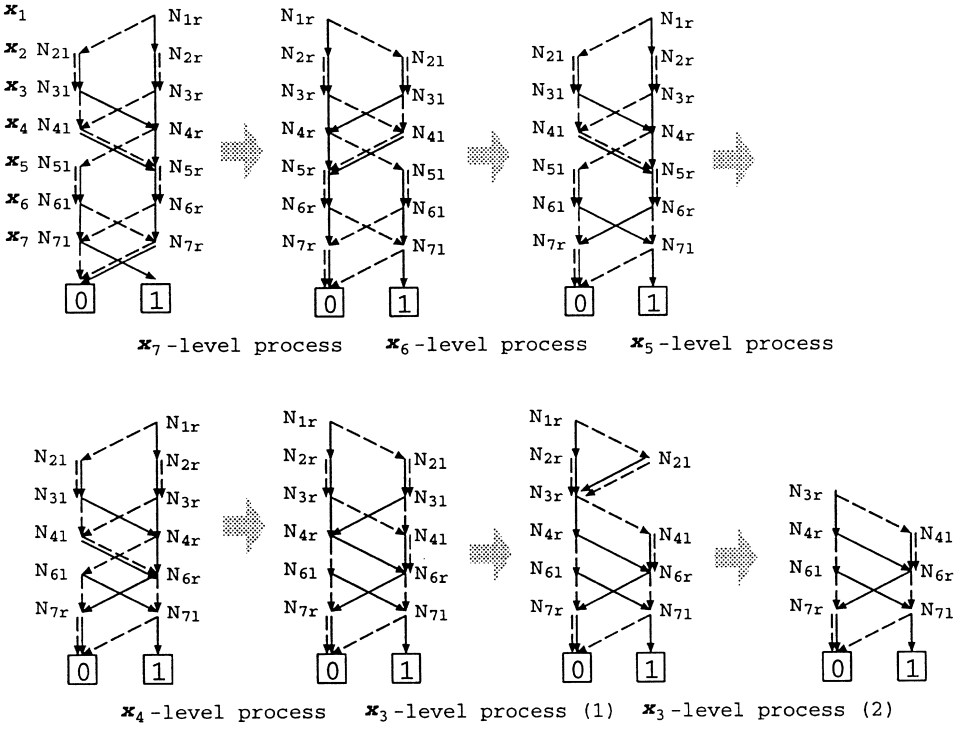


Fig. 4. Application of Body-Conversion.

1. If L_l is the level of sink nodes, stop.
2. If L_{l-1} is an x-type or the level of sink nodes ($l=1$), place the root node on the same lane as the node in L_{l-1} which is pointed to by the 0-labelled edge outgoing from the root node.
3. If L_{l-1} is an a-v-type ($a=0$ or 1), place the root node on the $\overline{a}$-lane.

Fig. 5. Procedure Root-Conversion.

In order to represent these two OBDDs by a unique representation, it is necessary to convert them into a layer list.

Consider the four outgoing edges from a v-type level. Any two edges having the same value and pointing to the same node are referred to as *merging edges*. Others are called *non-merging edges*. Additionally, we refer to two outgoing edges from a branching node as *branching edges* and those from a non-branching node as *non-branching edges* (see Fig. 8).

We define the x-level label in a normal-form width-2 OBDD as follows: when the x-level is a v-type, its label is defined as literal x if the label of its merging edges is 1, and as literal $\overline{x}$ otherwise; when the x-level is an x-type, its label is defined as literal x.

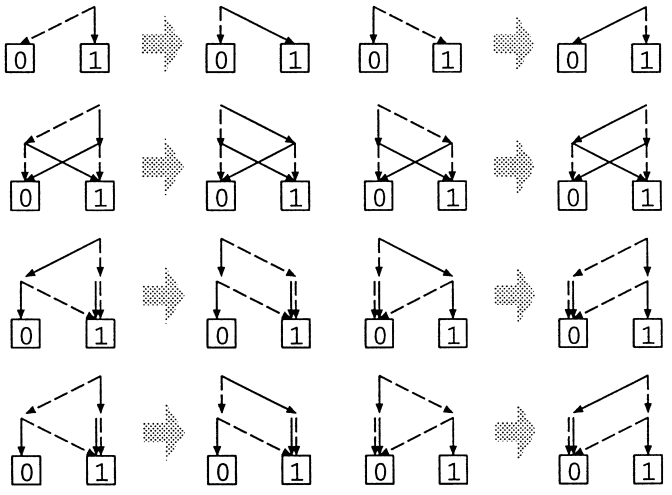


Fig. 6. Some applications of Root-Conversion.

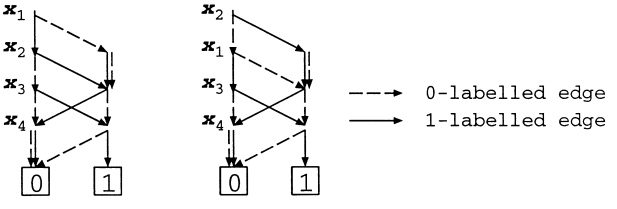


Fig. 7. Different normal-form width-2 OBDDs representing the same function.

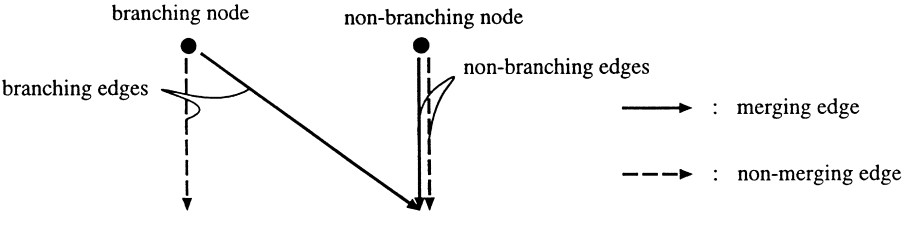


Fig. 8. Four outgoing edges from a v-type level.

A *layer* is a part of D composed of consecutive levels of the same type. We define the type of a layer to be the type of levels in the layer. A *layer list* of a normal-form width-2 OBDD is a list of triples $[(\emptyset, X_0, a_0), (V_1, X_1, a_1), \ldots, (V_d, X_d, a_d)]$, where, V_i is the set of labels of the levels in the ith v-type layer from the bottom of the OBDD; X_i is the set of labels of the levels in the x-type layer between the ith v-type layer and the $(i+1)$th v-type layer; and a_i is 1 if the $(i+1)$th v-type layer is a 1–v-type, and a_i is 0 otherwise. Note that we define X_0 and X_d as the sets of labels of the levels in the

x-type layer below V_1 and above V_d, respectively, and that a_d is defined to be 1 if the root node is on the 1-lane, and 0 otherwise. Now the question is which layer the level of the root node should belong to ? Deciding the type of the root-node level resolves this question. The root-node level is defined to be the same type as the previous level when there are more than one level in the OBDD except the level of the sink nodes, and it is defined to be an x-type if there is only one level. Note that each X_i can be empty. We say that d is the *length* of the layer list.

The two normal-form width-2 OBDDs in Fig. 7 are represented by the same layer list $[(\emptyset,\emptyset,0)(\{\overline{x_4}\},\{x_3\},1)(\{\overline{x_1},x_2\},\emptyset,0)]$.

3.1.3. Representation uniqueness

Theorem 2. *Width-2 OBDDs D_1 and D_2 represent the same function if and only if their respective layer lists coincide.*

Proof. Without loss of generality, we can assume that D_1 and D_2 are in normal form.

The 'if' part is trivial because, if the layer lists of D_1 and D_2 coincide, the difference between them must be their variable ordering in each layer, and changes in the variable ordering in any layer will not result in any changes in the function.

Let us prove the 'only if' part. Let f be the function represented by D_1 and D_2 and let $[(V_0^1,X_0^1,a_0^1),\ldots,(V_{d_1}^1,X_{d_1}^1,a_{d_1}^1)]$ and $[(V_0^2,X_0^2,a_0^2),\ldots,(V_{d_2}^2,X_{d_2}^2,a_{d_2}^2)]$ be the layer lists of D_1 and D_2, respectively. Assuming that $(V_j^1,X_j^1,a_j^1)=(V_j^2,X_j^2,a_j^2)$ for all $j<i$, let us prove $(V_i^1,X_i^1,a_i^1)=(V_i^2,X_i^2,a_i^2)$. Note that the following proof is applicable even when $i=0$.

First, we prove $V_i^1=V_i^2$. By the definition of a layer list, $V_0^1=V_0^2=\emptyset$. Assume that $V_i^1\neq V_i^2$ for $i>0$. Without loss of generality, we can assume that $V_i^1-V_i^2\neq\emptyset$. Let $x^a\in V_i^1-V_i^2$. There exist variable y and $b\in\{0,1\}$ such that x and y are different variables and $y^b\in V_i^2\cup X_i^2\cup V_{i+1}^2$ holds by the following reason. If such y^b does not exist, $V_i^2=\emptyset$ or $(V_i^2=\{x^{\bar{a}}\}$ and $X_i^2\cup V_{i+1}^2=\emptyset)$. The condition that $V_i^2=\emptyset$ means $d_2=i-1$, and implies that D_2 does not have x-level, which contradicts the fact that x is a relevant variable of f. When $V_i^2=\{x^{\bar{a}}\}$, it always holds that $X_i^2\cup V_{i+1}^2\neq\emptyset$, namely, the ith v-type layer is not the highest layer, because, if the highest layer is a v-type, it must have at least two levels by the definition of the type of the root-node level. Thus, there exists a variable y which differs from x and satisfies $y^b\in V_i^2\cup X_i^2\cup V_{i+1}^2$ for $b=0$ or 1. Then, there exists an assignment e such that $x(e)=a$, and Path (D_2,e) contains the branching non-merging outgoing edges from z-levels for all z satisfying z or $\bar{z}\in V_k^2$ and contains no merging outgoing edges from z-levels for any z satisfying z or $\bar{z}\in V_j^2$ for any $j<k$, where k is equal to i when $y^b\in V_i^2\cup X_i^2$ and $i+1$ when $y^b\in V_{i+1}^2$. For this assignment e, Path(D_1,e) and Path $(D_1,e_{\bar{y}})$ reach the same sink node, but Path(D_2,e) and Path$(D_2,e_{\bar{y}})$ reach the different sink nodes. (See Fig. 9.) This contradicts the assumption that D_1 and D_2 are representations of the same function.

Next, let us prove $X_i^1=X_i^2$. Assume that $X_i^1\neq X_i^2$. Without loss of generality, we can assume that $x\in X_i^1-X_i^2$. Since x is relevant to the function represented by D_2,

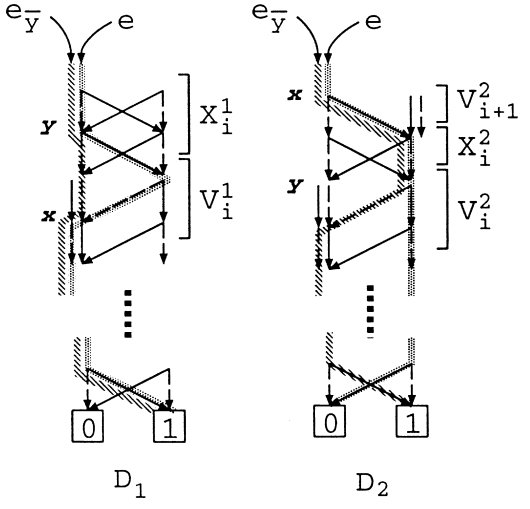


Fig. 9. Path(D_1, e), Path$(D_1, e_{\bar{y}})$,Path(D_2, e) and Path$(D_2, e_{\bar{y}})$.

x-level is contained in the kth v- or x-type layer ($k \geqslant i+1$). This implies that $V_{i+1}^2 \neq \emptyset$. Let y^b be an element in V_{i+1}^2. Then, there exists an assignment e such that Path(D_2, e) contains the non-branching merging outgoing edge from the y-level and contains no merging outgoing edges from z-levels for any z satisfying z or $\bar{z} \in V_j^2$ for any $j \leqslant i$. For this assignment e, Path(D_1, e) and Path$(D_1, e_{\bar{x}})$ reach the different sink nodes, but Path(D_2, e) and Path$(D_2, e_{\bar{x}})$ reach the same sink node. (See Fig. 10.) This contradicts the assumption that D_1 and D_2 are representations of the same function.

Finally, we show $a_i^1 = a_i^2$ on the assumption that $(V_j^1, X_j^1, a_j^1) = (V_j^2, X_j^2, a_j^2)$ for all $j < i$, $V_i^1 = V_i^2$ and $X_i^1 = X_i^2$. If $i = d_1$, $i = d_2$ must hold. In this case, $a_i^1 = a_j^1$ holds by the type definition of the root-node level and by the assumption that D_1 and D_2 represent the same function. Consider the case when $V_{i+1}^1 \neq \emptyset$. Assume that $a_i^1 \neq a_i^2$. Let $x^a \in V_{i+1}^1$. Then, $V_{i+1}^2 \neq \emptyset$ must hold. Note that $x^a \notin V_{i+1}^2$, because, if $x^a \in V_{i+1}^2$, there exists an assignment e with $x(e) = a$ such that Path(D_1, e) and Path(D_2, e) reach the different sink nodes. Let y^b be an element in V_{i+1}^2 if $V_{i+1}^2 \neq \{x^{\bar{a}}\}$, and an element in $X_{i+1}^2 \cup V_{i+2}^2$ if $V_{i+1}^2 = \{x^{\bar{a}}\}$. Then, there exists an assignment e with $x(e) = a$ such that Path(D_2, e) and Path$(D_2, e_{\bar{y}})$ reach the different sink nodes, which contradicts the assumption that D_1 and D_2 represent the same function, because Path(D_1, e) and Path$(D_1, e_{\bar{y}})$ reach the same sink node. Therefore, $a_i^1 = a_i^2$. □

3.2. Decision list representation

A *linear function* on X is a function representable by the form $f(x_1, \ldots, x_n) = \sum_{i=1}^n a_i x_i + a_0$, using $(a_0, \ldots, a_n) \in \{0,1\}^{n+1}$, where the addition and multiplication are operations on GF(2). Let $\mathscr{LF}$ be the class of linear functions on X and $\mathscr{LF}_k$ denote the subclass of $\mathscr{LF}$ composed of functions having at most k non-zero coefficients

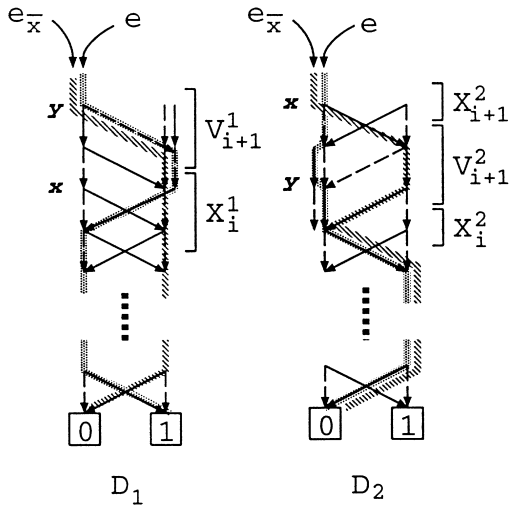


Fig. 10. Path(D_1, e), Path($D_1, e_{\overline{x}}$),Path(D_2, e) and Path($D_2, e_{\overline{x}}$).

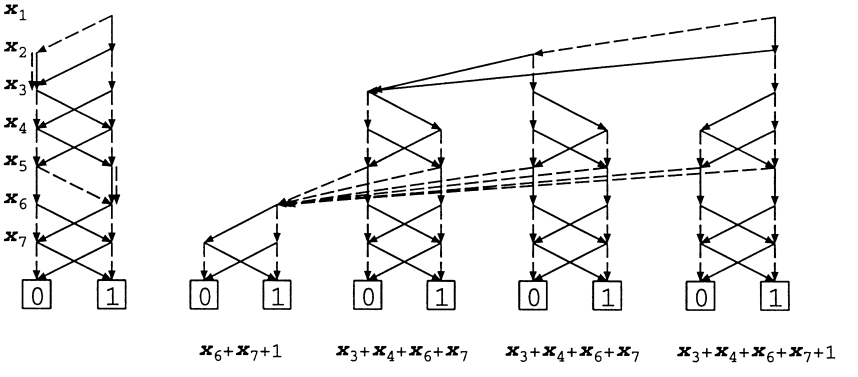


Fig. 11. The left width-2 OBDD is represented by a decision list in $(\mathcal{LF}_1, \mathcal{LF})$-$\mathcal{DL}$.

in $\{a_1, \ldots, a_n\}$. Every width-2 OBDD belongs to the class $(\mathcal{LF}_1, \mathcal{LF})$-$\mathcal{DL}$ [7, 6], whose intuitive explanation is given in Fig. 11.

Let D be a normal-form width-2 OBDD represented by layer list $[(\emptyset, X_0, a_0), \ldots, (V_d, X_d, a_d)]$. A corresponding decision list $L_D \in (\mathcal{LF}_1, \mathcal{LF})$-$\mathcal{DL}$ is constructed as follows. For each $y^b \in V_j$ for all $j \in \{1, \ldots, d\}$, let

$$I_{y^b} = \left(y + \overline{b}, \sum_{x \in \bigcup_{i<j} X_i} x + a_{j-1} \right).$$

Then, L_D is a sequence of I_{y^b} starting with all $y^b \in V_1$, followed by all $y^b \in V_2, \ldots$, followed by all $y^b \in V_d$ and ending with the item

$$\left(\mathbf{1}, \sum_{x \in \bigcup_{i \leqslant d} X_i} x + a_d \right).$$

Note that L_D has d alternations and that the alternation level of I_{y^b} for $y^b \in V_j$ is $j-1$.

4. Learning via equivalence queries alone

In this section, we present an algorithm which exactly learns width-2 OBDDs from equivalence queries alone. Here we consider the more general problem of learning $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$, where $\mathscr{F}$ is a finite function class and $\mathscr{G}$ is a function class which is learnable using equivalence queries alone. Note that the class of width-2 OBDDs is a subclass of $(\mathscr{LF}_1, \mathscr{LF})$-$\mathscr{DL}$.

Recently, Bergadano et al. [4] have independently shown an elegant result that the class of width-2 branching programs with a bounded number of sinks, a superclass of width-2 OBDDs, is learnable via equivalence queries alone. Their result concerning learnability of branching programs is more general than ours. However, our algorithm has the following merits. Our algorithm can learn $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$ for any finite function class $\mathscr{F}$ and any learnable class $\mathscr{G}$ via equivalence queries alone. Besides, if $\mathscr{G}$ is proper learnable, then $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$ is also proper learnable by using our algorithm.

We apply Simon's algorithm [10], which learns rank-r decision trees using equivalence queries, to the problem of learning $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$. Although $(\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$ is a class of rank-2 decision trees over base $\mathscr{F} \cup \mathscr{G}$, we cannot apply the algorithm as is because it is not applicable when the cardinality of $\mathscr{G}$ is not polynomial. But this problem can be overcome with a slight modification of the algorithm when $\mathscr{G}$ is learnable.

Let $A_{\mathscr{G}}$ be an exact learning algorithm for class $\mathscr{G}$ using at most m proper equivalence queries. Our algorithm 'G-DECLIST($A_{\mathscr{G}}$)', which is derived from Simon's 'DECTREE' algorithm by replacing each recursive call to itself with a call to $A_{\mathscr{G}}$, keeps a hierarchical partition $\{F_0, F_1, \ldots, F_N\}$ of $\mathscr{F}$, where N is the cardinality of $\mathscr{F}$. Initially, $F_0 = \mathscr{F}$ and $F_1 = \cdots = F_N = \emptyset$. Let S be the set of labelled counterexamples given so far. G-DECLIST constructs $S_0, \ldots, S_N$ as follows:

$$S_0 = S, S_{i+1} = \{(\boldsymbol{x}, b) \in S_i : \forall f \in F_i[f(\boldsymbol{x}) = 0]\}.$$

G-DECLIST executes $A_{\mathscr{G}}$ for each $f \in \mathscr{F}$. Assume that each $A_{\mathscr{G}}$ for f is waiting for the answer to an equivalence query with hypothesis g_f. Initially, g_f is the hypothesis of the first equivalence query asked by $A_{\mathscr{G}}$. Define I_f to be (f, g_f). Then, the hypothesis decision list L produced by G-DECLIST is a list of I_f starting with all $f \in F_0$, followed by $f \in F_1, \ldots, f \in F_N$.

Given a counterexample $\boldsymbol{x}$, G-DECLIST updates its hypothesis decision list as follows. Let b be the correct label of $\boldsymbol{x}$. Let $i_x = \min\{i : \exists f \in F_i[f(\boldsymbol{x}) = 1]\}$. Add $(\boldsymbol{x}, b)$ to

S_i for all $i \leqslant i_x$, and execute procedure UPDATE-LEVEL$((\boldsymbol{x}, b), i_x)$. UPDATE-LEVEL $((\boldsymbol{x}, b), i)$ is the following procedure. Let $F_i^x = \{f \in F_i : f(\boldsymbol{x}) = 1, g(\boldsymbol{x}) \neq b\}$. For all $f \in F_i^x$, UPDATE-LEVEL executes procedure UPDATE-g_f or procedure CHANGE-LEVEL depending on how many equivalence queries $A_{\mathscr{G}}$ for f has asked so far. If $A_{\mathscr{G}}$ for f has not asked m equivalence queries yet, UPDATE-LEVEL executes UPDATE-g_f, which gives counterexample $(\boldsymbol{x}, b)$ to $A_{\mathscr{G}}$ and updates g_f to the hypothesis of the next equivalence query asked by $A_{\mathscr{G}}$. If $A_{\mathscr{G}}$ for f has already asked m equivalence queries, UPDATE-LEVEL executes CHANGE-LEVEL. CHANGE-LEVEL moves f from F_i to F_{i+1}, resets $A_{\mathscr{G}}$ for f and restarts it. Let $T_f = \{(\boldsymbol{x}, b) \in S_{i+1} : f(\boldsymbol{x} = 1\}$. CHANGE-LEVEL executes $A_{\mathscr{G}}$ by returning an inconsistent member in T_f as a counterexample until hypothesis g_f of an equivalence query asked by $A_{\mathscr{G}}$ becomes consistent with T_f. CHANGE-LEVEL also moves $(\boldsymbol{x}', b')$ satisfying $f'(\boldsymbol{x}') = 0$ for all $f' \in F_i$ from S_i to S_{i+1}, and execute UPDATE-LEVEL$((\boldsymbol{x}', b'), i + 1)$ for all moved $(\boldsymbol{x}', b')$.

The following lemma guarantees that the number of f's moving from F_i to F_{i+1} for some i is at most its alternation level in the target decision list. We omit the proof of this lemma because the proof of Lemma 3 in [10] is available even for this lemma.

Lemma 3 (Cf. Lemma 3 in Simon [10]). *For $f \in \mathscr{F}$, let $l(f)$ be the integer i satisfying $f \in F_i$ during the run of G-DECLIST$(A_{\mathscr{G}})$ and $a^*(f)$ be the alternation level of f in the target list L^*. Then $l(f) \leqslant a^*(f)$.*

Theorem 4 in [10] is easily generalized to the next theorem, whose bound on the number of queries coincides with that of Theorem 4 in [10] when $m = 1$.

Theorem 4. *Let $\mathscr{F}, \mathscr{G}$ be classes of Boolean functions on X and $|\mathscr{F}| = N$. Assume the existence of exact learning algorithm $A_{\mathscr{G}}$ for the class $\mathscr{G}$ from at most m proper equivalence queries. Then G-DECLIST$(A_{\mathscr{G}})$ exactly learns $L^* \in (\mathscr{F}, \mathscr{G})$-$\mathscr{DL}$ of length d with l alternations from at most*

$$mN + (m+1)(Nl - \tfrac{1}{2}l^2 + \tfrac{1}{2}l - d) \leqslant mN + (m+1)(Nd - \tfrac{1}{2}d^2 - \tfrac{1}{2}d)$$
$$\leqslant \tfrac{1}{2}(m+1)N^2 + \tfrac{1}{2}(m-1)N$$

proper equivalence queries.

Proof. The latter two bounds independent of l are easily obtained[5] by using the fact that $l \leqslant d \leqslant N$. Note that $A_{\mathscr{G}}$ is executed for each $f \in \mathscr{F}$. Whenever G-DECLIST$(A_{\mathscr{G}})$ gets a counterexample, at least one $A_{\mathscr{G}}$ gets a counterexample. The algorithm $A_{\mathscr{G}}$ for $f \in \mathscr{F}$ gets at most $m + 1$ counterexamples when $f \in F_i$ for each $i < a^*(f)$, at most m when $f \in F_{a^*(f)}$, and because of Lemma 3, it gets no more counterexamples, where

[5] Note that

$$Nl - \tfrac{1}{2}l^2 = Nl + (d-l)\tfrac{d+l}{2} - \tfrac{1}{2}d^2 \leqslant Nl + (d-l)N - \tfrac{1}{2}d^2 = Nd - \tfrac{1}{2}d^2.$$

$a^*(f)$ is the alternation level of f in L^*. Thus at most $(m+1)a^*(f)+m$ counterexamples are necessary for each $f \in \mathscr{F}$. In the worst case, L^* has only one function f with $a^*(f)=i$ for each $i \in \{0,1,\ldots,l-2\}$, $d-l+1$ functions f with $a^*(f)=l-1$, and $N-d$ functions f with $a^*(f)=l$. Therefore, the total number of counterexamples in the worst case is

$$\begin{aligned}&\sum_{f\in\mathscr{F}}((m+1)a^*(f)+m)\\ &\quad\leqslant mN+(m+1)(0+1+\cdots+(l-2)+(d-l+1)(l-1)+(N-d)l)\\ &\quad= mN+(m+1)(Nl-\tfrac{1}{2}l^2+\tfrac{1}{2}l-d).\end{aligned}$$ □

Lemma 5. *There is a polynomial-time algorithm which exactly learns $\mathscr{LF}$ with at most $n+1$ proper equivalence queries.*

Proof. Consider an algorithm which asks an equivalence query with a hypothesis linear function that is consistent with all the counterexamples so far. Finding such a linear function is done by solving linear equations with $n+1$ variables. Since adding the linear equation for a new counterexample reduces the degree of freedom by one, $n+1$ counterexamples are enough to find a unique, consistent linear function. □

Corollary 6. *There is a polynomial-time algorithm which exactly learns an arbitrary function in $(\mathscr{LF}_k, \mathscr{LF})$-$\mathscr{DL}$ with l alternations using $O(n^{k+1}l)$ proper equivalence queries.*

Proof. Let $A_{\mathscr{LF}}$ denote an exact learning algorithm for $\mathscr{LF}$ which uses at most $n+1$ proper equivalence queries and whose existence is guaranteed by Lemma 5. Since $|\mathscr{LF}_k| = O(n^k)$, we can obtain this corollary by applying Theorem 4 to the case with $\mathscr{F}=\mathscr{LF}_k$ and $\mathscr{G}=\mathscr{LF}$. □

Bshouty et al. [6] noted that $(\mathscr{LF}_2, \mathscr{LF})$-$\mathscr{DL}$ is equivalent to the class of *strict width-2* branching programs and that $(\mathscr{LF}_1, \mathscr{LF})$-$\mathscr{DL}$ is equivalent to the class of *levelled strict width-2* branching programs. They showed proper PAC learnability of these two subclasses of branching programs by using these relationships and polynomial-time convertibility. Here, by making use of these relationships, as well as of polynomial-time convertibility and of Corollary 6, we can also show exact learnability of these subclasses from proper equivalence queries alone. Note that learnability via proper equivalence queries is stronger than proper PAC learnability.

The next corollary describes our result on learning width-2 OBDDs via equivalence queries alone.

Corollary 7. *There exists a polynomial-time algorithm which exactly learns an arbitrary width-2 OBDD D using $O(n^2l)$ equivalence queries, where l is the length of the layer list of D.*

5. Learning via proper equivalence and membership queries

In this section, we present an algorithm which learns width-2 OBDDs using proper equivalence queries and membership queries.

Let $[(\emptyset, X_0, a_0), (V_1, X_1, a_1), \ldots, (V_d, X_d, a_d)]$ be a layer list of a width-2 OBDD D. For $0 \leqslant i < d$, we call $[(\emptyset, X_0, a_0), (V_1, X_1, a_1), \ldots, (V_i, X_i, a_i)]$ a *layer sublist* of D and let D_i denote the OBDD represented by it. When $X_0 \neq \emptyset$, we also include $[(\emptyset, \emptyset, 1)]$ and $[(\emptyset, \emptyset, 0)]$ in the set of layer sublists. Consider the case that the layer list of D is $[(\emptyset, \{x_2, x_3\}, 1), (\{\overline{x_1}\}, \emptyset, 0), (\{\overline{x_4}, x_5\}, \emptyset, 1)]$. There are four layer sublists of D: $[(\emptyset, \{x_2, x_3\}, 1), (\{\overline{x_1}\}, \emptyset, 0)]$, $[(\emptyset, \{x_2, x_3\}, 1)]$, $[(\emptyset, \emptyset, 0)]$ and $[(\emptyset, \emptyset, 1)]$. OBDDs corresponding to these four layer sublists are shown in Fig. 12. Note that,[6] for any assignment e, $D(e) = D_i(e)$ if $\mathrm{Path}(D, e)$ contains the node which corresponding to the root node of D_i. Thus, in the case where D is the target OBDD and D_i is a hypothesis, $\mathrm{Path}(D, e)$ for a counterexample e never contains that node. Note that D_1 in Fig. 12 is not in normal form because the application procedure Root-Conversion makes the root node place on the other lane. (See Fig. 13.) Therefore, by the definition of layer lists, the layer list of D_1 is $[(\emptyset, \{x_1, x_2, x_3\}, 1)]$, which we call a *normalized layer sublist* of D. Thus, a normalized layer sublist of D is a layer list of the function represented by one of the layer sublists of D.

Algorithm W2-OBDD-QLearning has the initial hypothesis $[(\emptyset, \emptyset, 0)]$. Given a counterexample for the current hypothesis, the algorithm constructs a new hypothesis, which is a normalized layer sublist of the target width-2 OBDD and is longer than the current hypothesis. Thus, given at most n counterexamples, the hypothesis of the algorithm coincides with the layer list of the target OBDD.

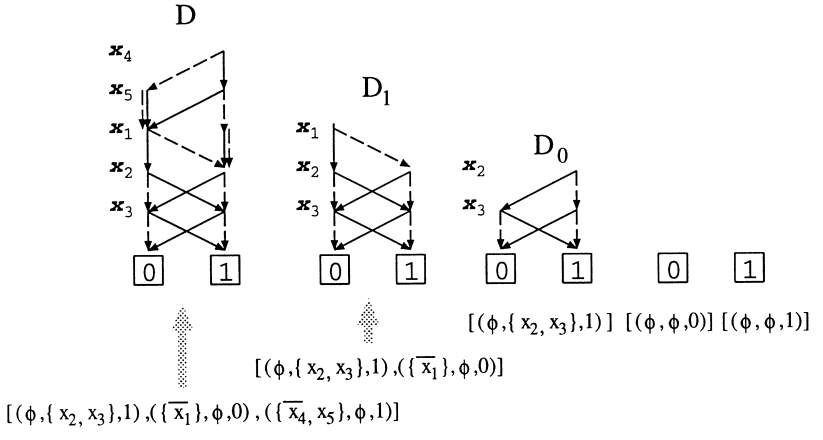


Fig. 12. OBDDs corresponding to layer sublists of D.

[6] We are somewhat loose in our notation here, using a representation of a function as if it were the function itself.

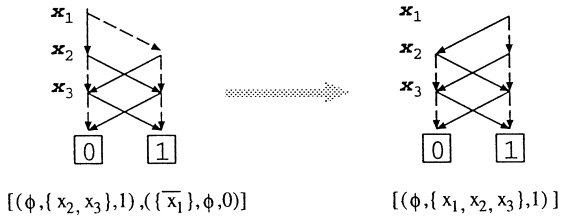


Fig. 13. Conversion into the normalized layer sublist.

Denormalization (H, e) return(H')
H : normalized layer sublist of D
e : counterexample for H
H' : layer sublist of D that is equivalent to H
Construct-normalized-sublist (H, e) return(H')
H : layer sublist of D or $[(\emptyset, \emptyset, 0)]$
e : counterexample for H
H' : the layer list of D when Path(D, e) contains no non-branching merging edge, and the layer list of D_i otherwise, where i is the minimum j satisfying the condition that Path(D, e) contains a non-branching merging edge in the $(j+1)$th v-type layer

Fig. 14. Two procedures used in W2-OBDD-QLearning.

Let D be the target width-2 OBDD in normal form. Assume the existence of the two procedures described in Fig. 14: *Denormalization* and *Construct-normalized-sublist*.

W2-OBDD-QLearning is described in Fig. 15. In Figs. 15, 16 and 19, $S(H, e)$ denotes set $\{x \in \text{unused-variables}(H) : D(e_{\bar{x}}) \neq D(e)\}$, where unused-variables$(H)$ is the set of variables which do not appear in H. The output (ANS,e) of EQUIV(H) is the answer returned to an equivalence query with hypothesis H, where ANS is YES or NO and e is a counterexample.

Lemma 8. *Assume the existence of the procedures Denormalization and Construct-normalized-sublist with inputs and outputs as in Fig.* 14. *Algorithm W2-OBDD-Q Learning then exactly learns an arbitrary width-2 OBDD.*

Proof. Let $(\text{ANS}_0, e_0) = \text{EQUIV}([(\emptyset, \emptyset, 0)])$. If the target OBDD represents the 0-valued constant function, then $\text{ANS}_0 = \text{YES}$. If the target represents the 1-valued constant function, then $\text{ANS}_0 = \text{NO}$, and EQUIV$([(\emptyset, \emptyset, 1)])$ is executed next because $S([(\emptyset, \emptyset, 0)], e_0) = \emptyset$, and YES is returned.

In the other cases, Denormalization and Construct-normalized-sublist are executed at least once, for the following reason. Let the target width-2 OBDD be $[(\emptyset, X_0, a_0), (V_1, X_1, a_1), \ldots, (V_d, X_d, a_d)]$. If $X_0 \neq \emptyset$, $X_0 \subseteq S([(\emptyset, \emptyset, 0)], e_0)$. If $X_0 = \emptyset$, $a_0 = 0$ and $d > 0$, then $\emptyset \neq V_1 \subseteq S([(\emptyset, \emptyset, 0)], e_0)$. If $X_0 = \emptyset$, $a_0 = 1$ and $d > 0$, then $\emptyset \neq V_1 \subseteq S([(\emptyset, \emptyset, 1)], e_1)$,

```
01 begin
02   H := [(∅, ∅, 0)]
03   repeat
04     (ANS, e) := EQUIV(H)
05     if ANS = YES then exit repeat
06     if S(H, e) = ∅ then H := [(∅, ∅, 1)]
07       else H := Denormalization (H, e)
08            H := Construct-normalized-sublist (H, e)
09   end repeat
10 end
```

Fig. 15. Procedure W2-OBDD-QLearning.

```
01 input: H: normalized layer sublist [(∅, X_0, a_0), ..., (V_i, X_i, a_i)] of D,
         e: counter example for H
02 output: H': layer sublist of D
03 begin
04     let x ∈ S(H, e).
05     if H = [(∅, ∅, 0)] or [(∅, ∅, 1)] then return H
06     if X_i ≠ ∅ then
07         X := {y ∈ X_i : D(e_ȳ) ≠ D(e)}
08         if X ≠ X_i then
09             let z ∈ X_i − X.
10             return
                 [(∅, X_0, a_0), ..., (V_{i−1}, X_{i−1}, a_{i−1}), (V_i, X, a_i^{z(e)}), ({z^{\overline{z(e)}}}, ∅, a_i^{\overline{z(e)}})]
11         else return H
12     else
13         X := {y^b ∈ V_i : D(e_ȳ) ≠ D(e), b = 0, 1}
14         if X = ∅ then return H
15         Y := {y^b ∈ X : D(e_{x̄,ȳ}) ≠ D(e_x̄), b = 0, 1}
16         if Y ≠ ∅ then
17             let z^a ∈ Y.
18             return [(∅, X_0, a_0), ..., (V_{i−1}, X_{i−1}, a_{i−1}), (V_i − Y, {z}, a_i^a)]
19         else
20             let z^a ∈ V_i − X.
21             return [(∅, X_0, a_0), ..., (V_{i−1}, X_{i−1}, a_{i−1}), (X, ∅, a_i), ({z^ā}, ∅, \overline{a_i})]
22 end
```

Fig. 16. Procedure Denormalization.

where e_1 is the counterexample returned by EQUIV($[(\emptyset,\emptyset,1)]$). Thus, in any case, the two subprocedures are executed with $H=[(\emptyset,\emptyset,0)]$ or $[(\emptyset,\emptyset,1)]$.

By definition of Denormalization and Construct-normalized-sublist, H is the layer list of D_i for some i after the first execution of the two subprocedures.

Assume that H is the layer list of D_i for some $i<d$ with which an equivalence query is asked. We prove that H becomes the layer list of $D_{i'}$ $(i'>i)$ or D when the next equivalence query is asked. Let e be a counterexample for H.

Path(D,e) contains no merging edge in the jth v-type layer for $j<i+1$ by the following reason. Let Path$_D(D_i,e)$ denote the path in D corresponding to Path(D_i,e). Path$_D(D_i,e)$ and Path(D,e) never contain the same node, because, if so, the two paths coincide from the node to one of the sink nodes, which contradicts the fact that e is a counterexample for D_i. If Path(D,e) contains a merging edge in the jth v-type layer for $j<i+1$, then Path$_D(D_i,e)$ also contains a merging edge at the same level, and it is implied that the two paths contains the same node.

Furthermore, in the $(i+1)$th v-type layer, Path(D,e) must contain the branching non-merging edges only, because e is a counterexample for D_i.

Therefore, $V_{i+1}\subseteq S(H,e)$ holds and the two subprocedures are executed. Thus, H output by Construct-normalized-sublist is $D_{i'}$ $(i'\geqslant i+1)$ or D. □

Lemma 9. *Let H' be a layer sublist of a width-2 OBDD D and H a normalized layer sublist of D that is equivalent to H'. Let e be a counterexample for H. Then, procedure Denormalization (H,e) given in Fig.* 16 *outputs H'.*

Proof. Let $H'=[(\emptyset,X'_0,a'_0),\ldots,(V'_j,X'_j,a'_j)]$. Note that $H'\neq H$ only when $|X'_j|=1$ or $(X'_j=\emptyset$ and $|V'_j|=1)$.

Let $H=[(\emptyset,X_0,a_0),\ldots,(V_i,X_i,a_i)]$.

First, consider the case with $X_i\neq\emptyset$. In this case, $H=H'$ or $(j=i+1,X'_j=\emptyset$ and $|V'_j|=1)$. Note that Path(D,e) contains no merging edge in the kth v-type layer in D for $k\leqslant j$ because e is a counterexample for H that is equivalent to H'. Thus, if $H=H'$, then $X\triangleq\{y\in X_i\colon D(e_{\overline{y}})\neq D(e)\}=X_i$ and the algorithm outputs H (Line 11, Fig. 17(1)).

Assume that $j=i+1$, $X'_j=\emptyset$ and $|V'_j|=1$. Let $V'_j=\{z^a\}$. Then, $a'_j=a_i^a$ and $a'_{j-1}=a_i^{\overline{a}}$. Since Path$(D,e)$ contains a non-branching non-merging edge at z-level, we obtain that $\{z\}=X_i-X$, $a=\overline{z(e)}$ and $X=X'_{j-1}$. Thus, the algorithm outputs H' (Line 10, Fig. 17(2)).

Next, consider the case with $X_i=\emptyset$. In this case, $H=H'$ or $|X'_j|=1$ or $(X'_{j-1}=X'_j=\emptyset$ and $|V'_j|=1)$. Consider the case with $H=H'$. Since Path(D,e) contains non-branching non-merging edges in the jth v-type layer, we obtain that $X\triangleq\{y^b\in V_i\colon D(e_{\overline{y}})\neq D(e),\ b=0,1\}=\emptyset$. Thus, the algorithm outputs H when $H=H'$ (Line 14, Fig. 18(1)).

Assume that $|X'_j|=1$. Let $X'_j=\{z\}$. Define a as 1 if $a'_j\neq a'_{j-1}$ and as 0 otherwise. Then $a_i=a'^a_j$ and $z^a\in V_i$. Since, in the z-level, Path(D,e) and Path$(D,e_{\overline{x}})$ contain different nodes, we obtain that $Y\triangleq\{y^b\in V_i\colon D(e_{\overline{y}})\neq D(e),\ D(e_{\overline{x},\overline{y}})\neq D(e_{\overline{x}}),\ b=0,1\}=\{z^a\}$. Therefore, the algorithm outputs H' (Line 18, Fig. 18(2)).

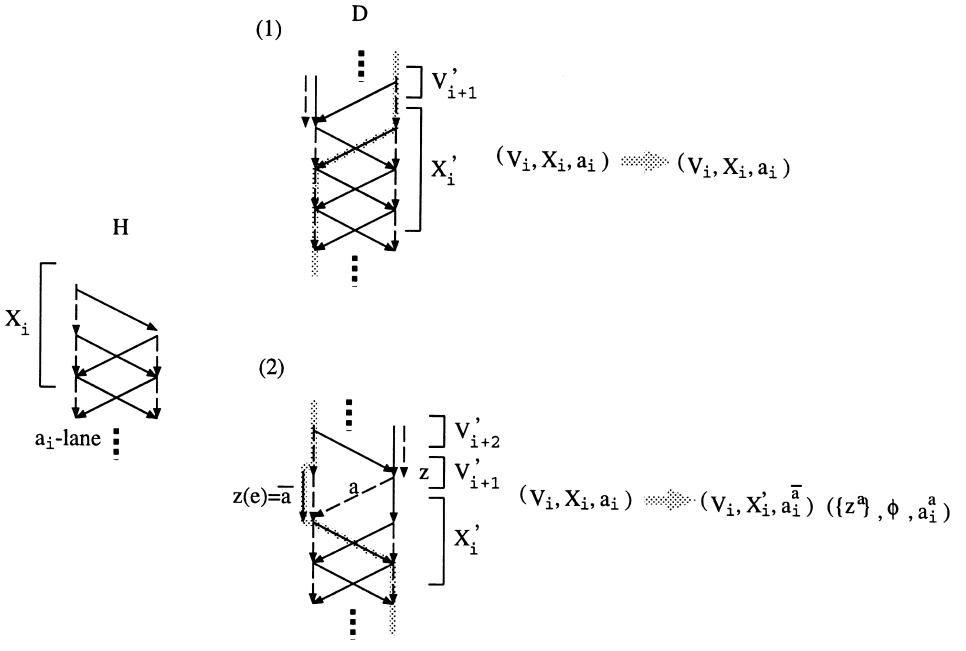


Fig. 17. Case with $X_i \neq \emptyset$.

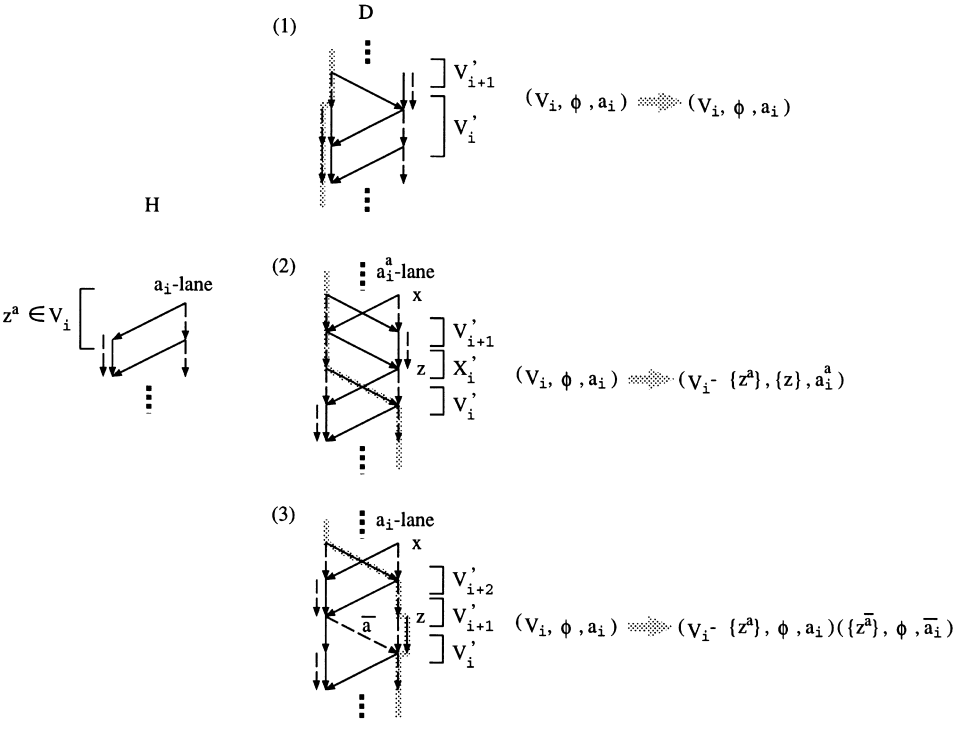


Fig. 18. Case with $X_i = \emptyset$.

Assume that $X'_{j-1}=X'_j=\emptyset$ and $|V'_j|=1$. Let $V'_j=\{z^a\}$. Then, $z^{\overline{a}}\in V_i$. For Y at Line 15 and X at Line 13, $Y=\emptyset$ and $X=V'_{j-1}$ hold because Path(D,e) contains a non-branching edge in the jth v-type layer, contains branching edges in the $(j-1)$th v-type layer, and contains no merging edges in the kth v-type layer for $k\leqslant j$. Thus, $V_i-X=\{z^{\overline{a}}\}$. Note that $a_i=\overline{a'_j}$. Therefore, the algorithm outputs H' (Line 21, Fig. 18(3)). □

Next, let us prove that the procedure Construct-normalized-sublist given in Figs. 19 and 20 is the kind of algorithm that is assumed to exist in Lemma 8. First, let us explain two functions used in the procedure: *Classify-and-sort* and *Append*. Let V be a finite set. Assume that set S_x is defined for each $x\in V$, and that $S_x\subseteq S_y$ or $S_y\subseteq S_x$ for any $x, y\in V$. Let k be the number of different elements in $\{S_x: x\in V\}$. Consider sets $S_1,\ldots,S_k$ which are k different sets in $\{S_x: x\in V\}$ satisfying $S_1\subset S_2\subset\cdots\subset S_k$. Then, function Classify-and-sort $(\{(x,S_x): x\in V\})$ outputs $(k,(S_1,V_1),\ldots,(S_k,V_k))$ which satisfies the condition that $\{V_1,\ldots,V_k\}$ is a partition of V such that $x\in V_i$ if and only if $S_x=S_i$. Function Append $((A_1,\ldots,A_{l_A}),B_1,\ldots,B_{l_B})$ is defined as $(A_1,\ldots,A_{l_A},B_1,\ldots,B_{l_B})$.

Now, we explain how procedure Construct-normalized-sublist finds the layers of a target OBDD and finds out their types and positions. In the following, we explain one easy case using an easy example for intuitive understanding. Let D be the normal-form width-2 OBDD shown in Fig. 21, which is represented by layer list $[(\emptyset,X_0^*,0),(V_1^*,X_1^*,0),(V_2^*,X_2^*,1),(V_3^*,\emptyset,0),(V_4^*,\emptyset,1),(V_5^*,\emptyset,0)]$. Let e be the assignment for which Path(D,e) is the one shown in Fig. 21. Note that Path(D,e) contains no merging edges, which is an important assumption here. According to whether Path(D,e) contains the branching or non-branching edge outgoing from an x-level, we can classify all the variables x in the v-type layers into two sets B and N: x is in B if branching and x is N if non-branching. Note that all the variables belonging to the same v-type layer are included in the same set (B or N). Let X denote the set of variables whose level is an x-type. For D in Fig. 21, $B=V_2^{*\prime}\cup V_3^{*\prime}\cup V_5^{*\prime}$, $N=V_1^{*\prime}\cup V_4^{*\prime}$ and $X=X_0^*\cup X_1^*\cup X_2^*$, where $V_i^{*\prime}$ is the set of variables x satisfying that x or $\overline{x}$ is in V_i^*.

Let $S=\{x: D(e)\neq D(e_{\overline{x}})\}$. Then, S is $B\cup X$. Thus, we can find all the variables in $B\cup X$ using membership queries. For $x\in S$, the procedure constructs $S_x=\{y\in S-\{x\}: D(e_{\overline{x}})\neq D(e_{\overline{x},\overline{y}})\}$. We can divide S into B and X because $x\in B$ if and only if there exists a different element $y\in S$ such that $S_x\subseteq S_y$. (The details are in the proof of Lemma 10.) Assume that $x\in V_i^{*\prime}\subseteq B$. Note that, at each level below the layer of X_i^*, the node contained in Path$(D,e_{\overline{x}})$ is different from the node contained in Path(D,e). Thus, at the layers of $V_j^{*\prime}\subseteq B$ for $j<i$, Path$(D,e_{\overline{x}})$ contains non-branching edges only. This means that no elements in $V_j^{*\prime}\subseteq B$ for $j<i$ is included in S_x. Furthermore, Path$(D,e_{\overline{x}})$ contains a merging edge at x-level. Thus, any assignment changes in the layer of V_i and above it do not affect the values of D. This means that no elements in X_j^* and $V_j^{*\prime}$ for $j\geqslant i$ is included in S_x. Therefore, $S_x=\bigcup_{j<i}X_j^*$. In Fig. 21, B is divided into two sets; $V_1^e=\{x: S_x=X_0^*\cup X_1^*\}$ and $V_2^e=\{x: S_x=X_0^*\cup X_1^*\cup X_2^*\}$. Set

```
01 input: H: layer sublist [(∅, X_0, a_0), ..., (V_l, X_l, a_l)] of D or [(∅, ∅, 0)]
          e: counterexample for H
02 output: H': normalized layer sublist of D or D
03 begin
   /*
   ** finding the set S of relevant variables which do not appear
   ** in hypothesis H and dealing with the case when only one level is found
   */
04    S := S(H, e), Z := unused-variables(H) − S
05    if |S| = 1 then
06       let x ∈ S
07       W_x := {y ∈ Z: D(e_{x̄,ȳ}) ≠ D(e_x̄)}
08       if W_x = ∅ then
09          if X_l ≠ ∅ or l = 0
10             then return [(∅, X_0, a_0), ..., (V_l, X_l ∪ {x}, ā_l ⊕ x(e))]
11             else return [(∅, X_0, a_0), ..., (V_l ∪ {x^{x(e)}}, ∅, a_l)]
12       else e := e_x̄, S := S ∪ W_x, Z := Z − W_x, a_l := ā_l
   /*
   ** dividing S into R_1, ..., R_h, where each R_i is composed of
   ** variables of at least one layer in D
   */
13    for each x ∈ S, S_x := {y ∈ S − {x}: D(e_{x̄,ȳ}) ≠ D(e_x̄)}
14    V := {x ∈ S: ∃y ∈ S s.t. x ≠ y, S_x ⊆ S_y}
15    (k, (S_1, V_1), ..., (S_k, V_k)) := Classify-and-sort({(x, S_x): x ∈ V})
16    S_0 := ∅
17    for i := 1 to k, (R_{2i−1}, t_{2i−1}) := (S_i − S_{i−1}, 'x'), (R_{2i}, t_{2i}) := (V_i, 'b')
18    (R_{2k+1}, t_{2k+1}) := (S − (V ∪ S_k), 'x')
19    if R_{2k+1} = ∅ then h := 2k else h := 2k + 1
```

Fig. 19. Procedure Construct-normalized-sublist (1).

X is also divided into two sets; $X_0^e = S_{x_1}$ and $X_1^e = S_{x_2} - S_{x_1}$ for $x_1 \in V_1^e$ and $x_2 \in V_2^e$. As a result, S is divided into four sets, X_0^e, V_1^e, X_1^e and V_2^e. The problem is that X_0^e and V_2^e are composed of variables from more than one layer, and the layers whose variables are in N are not yet found. Let Z be the set of variables which is not included in S. Let $W_x = \{y \in Z: D(e_{\bar{x}}) \neq D(e_{\bar{x},\bar{y}})\}$. Assume that $x \in X_i^*$. At the layers of $V_j^{*\prime} \subseteq N$ for $j > i$, Path$(D, e_{\bar{x}})$ contains non-branching edges only. On the other hand, at the layers of $V_j^{*\prime} \subseteq N$ for $j \leqslant i$, Path$(D, e_{\bar{x}})$ contains branching edges only. Thus, $W_x = \bigcup_{j \leqslant i,\, V_j^{*\prime} \subseteq N} V_j^{*\prime}$. This also holds when $x \in V_i^{*\prime} \subseteq B$. In Fig. 21, X_0^e is divided into two sets; $X_0 = \{x: W_x = \emptyset\}$ and set $X_1 = \{x: W_x = V_1^{*\prime}\}$. We can also find the v-type layer between them whose variable set is $W_{z_2} - W_{z_1}$ for $z_1 \in X_0$ and $z_2 \in X_1$.

```
   /*
   ** decomposing each set R_i into individual layers of D and finding
   ** v-type layers between them which contain non-branching edges in P(D, e)
   */
20    L := ∅
21    for i := 1 to h
22       if R_i ≠ ∅ then
23          for each x ∈ R_i, W_x := {y ∈ Z: D(e_{x̄,ȳ}) ≠ D(e_{x̄})}
24          (m, (W_1, U_1), ..., (W_m, U_m)) := Classify-and-sort({(x, W_x): x ∈ R_i})
25          L := Append(L, (W_1, 'n'), (U_1, t_i), (W_2 − W_1, 'n'), (U_2, t_i), ...,
                                                        (W_m − W_{m−1}, 'n'), (U_m, t_i))
26          Z := Z − W_m
   /*
   ** layer list construction
   */
27    let L = [(Y_1, y_1), ..., (Y_m, y_m)]
28    if |Y_m| = 1 then h := m − 2 else h := m
29    a := ā_l, j := l
30    for i := 1 to h
31       if y_i = 'x' then X_j := Y_i, a := a ⊕ ⨁_{x ∈ Y_i} x(e)
32       else V_{j+1} := {x^{\overline{x(e)}}: x ∈ Y_i}, X_{j+1} := ∅
33            if y_i = 'b' then a_j := ā else a_j := a
34            j := j + 1
35    H := [(∅, X_0, a_0), ..., (V_j, X_j, a_j)]
36    if |Y_m| = 1 then
37       let x ∈ Y_m
38       return Construct-normalized-sublist(H, e_{x̄})
39    else return H
40 end
```

Fig. 20. Procedure Construct-normalized-sublist (2).

Similarly, V_2^e is also divided into V_3', V_4' and V_5' which correspond to $V_3^{*\prime}, V_4^{*\prime}$ and $V_5^{*\prime}$, respectively. Therefore, we can find all layers of D and also their types and positions.

Construct-normalized-sublist consists of four parts. In the first part, that is from Lines 04 to 12, the procedure finds the set S of relevant variables which do not appear in hypothesis H and deals with the case when only one level is found. In the second part, that is from Lines 13 to 19, it divides S into $R_1, \ldots, R_h$, where each R_i is composed of variables of at least one layer in D. In the third part, that is from Lines 20 to 26, it decomposes each set R_i into individual layers of D and finds v-type layers between them, the layers containing non-branching edges in Path(D, e). All information needed

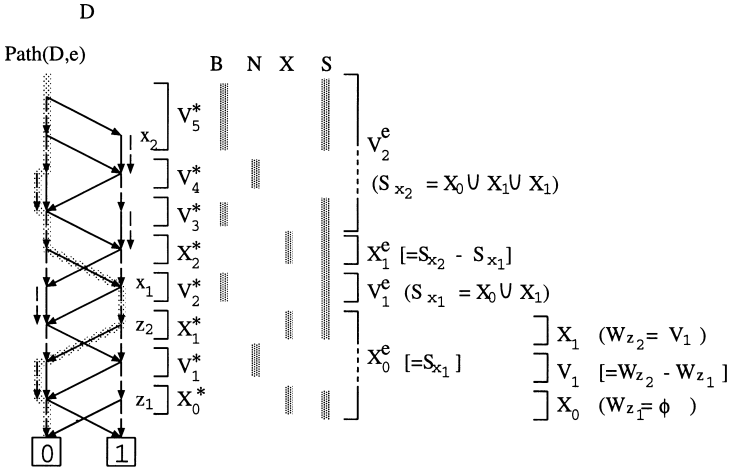


Fig. 21. Example of application of Construct-normalized-sublist.

to construct the layer list of D is obtained by the end of this part. The final part, from Lines 27 to the end of the procedure, constructs the layer list of D.

Lemma 10. *Let H be a layer sublist of the target width-2 OBDD D and e a counterexample for H. Then,*

(1) *the procedure Construct-normalized-sublist (H,e) given in Figs.* 19 *and* 20 *outputs the layer list of D when* Path(D,e) *contains no non-branching merging edge, and*

(2) *it outputs the layer list of D_i otherwise, where $i = \min\{j$: Path(D,e) contains non-branching merging edges in the $(j+1)$th v-type layer$\}$.*

Proof. Let $[(\emptyset, X_0^*, a_0^*), \ldots, (V_d^*, X_d^*, a_d^*)]$ be the layer list of D and $V_i^{*\prime}$ the set of variables x such that $x \in V_i^*$ or $\overline{x} \in V_i^*$. If Path(D,e) contains non-branching merging edges in the $(i+1)$th v-type layer, $D(e_{\overline{x}}) = D(e)$ for $x \in \bigcup_{j>i}(V_j^{*\prime} \cup X_j^*)$. Thus, in this case, the output of procedure Construct-normalized-sublist for the target OBDD D is the same as that for D_i because there is no difference between the answers for D and D_i to the all membership queries asked in the procedure. This fact allows us to focus on (1) of this lemma if H is also the layer sublist of D_i. The only case for which H, represented by $[(\emptyset, X_0, a_0), \ldots, (V_l, X_l, a_l)]$, is a layer sublist of D but not a layer sublist of D_i is when $l = i-1$, $X_i^* = \emptyset$ and $|V_i^*| = 1$. Let x be the unique variable in $V_i^{*\prime}$. In this case, Path(H,e) must contain the branching non-merging outgoing edge from the x-level because e is a counterexample for H. Thus, the layer list H' of D_i is $[(\emptyset, X_0^*, a_0^*), \ldots, (V_{i-1}^*, X_{i-1}^* \cup \{x\}, \overline{a_{i-1}^*} \oplus x(e))]$ if $X_{i-1}^* \neq \emptyset$ (Fig. 22(1)), and H' is $[(\emptyset, X_0^*, a_0^*), \ldots, (V_{i-1}^* \cup \{x^{x(e)}\}, \emptyset, a_{i-1}^*)]$ otherwise (Fig. 22(2)). In the procedure, $S = S(H,e) \triangleq \{x \in$ unused-variables(H): $D(e_{\overline{x}}) \neq D(e)\} = \{x\} = V_i^{*\prime}$ at Line 04, $|S| = 1$ at Line 05, and $W_x = \{y \in$ unused-variables$(H) - S$: $D(e_{\overline{x},\overline{y}}) \neq D(e_{\overline{x}})\} = \emptyset$ at Line 07

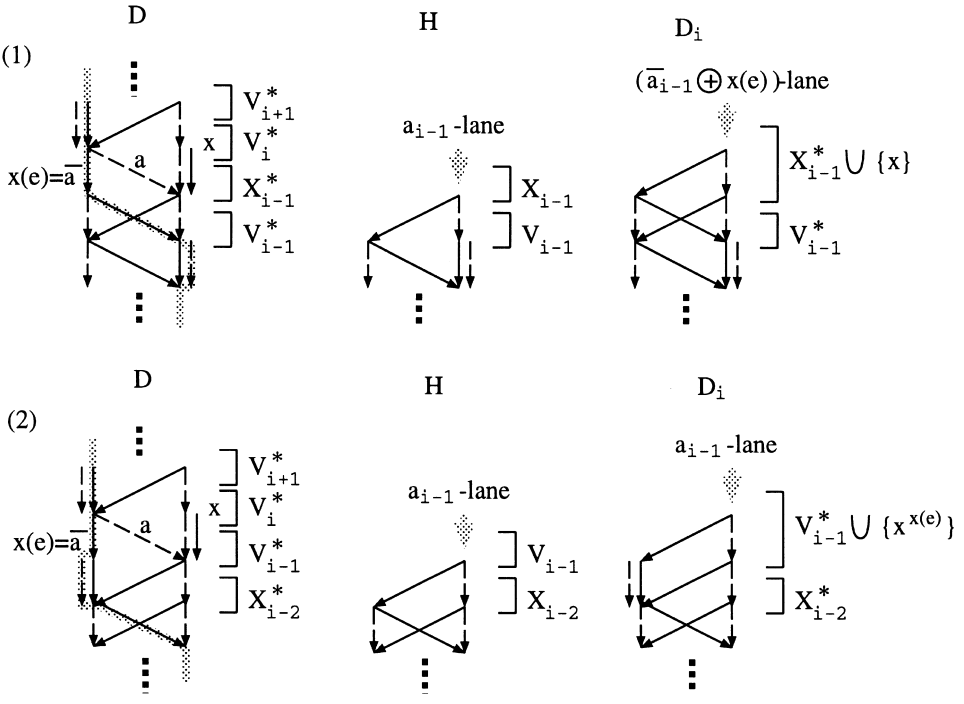


Fig. 22. Examples of H that is a layer sublist of D but not a layer sublist of D_i.

because Path$(D, e_{\bar{x}})$ contains a merging outgoing edge from x-level. Thus, the procedure outputs the layer list of D_i at Line 10 when $X^*_{i-1} \neq \emptyset$ and outputs it at Line 11 otherwise.

Let us now prove (1). We define $M(D,e)$ as follows:

$$M(D,e) \triangleq \{i\text{: There exists a (branching) merging edge in Path}(D,e) \text{ outgoing from a level in the } i\text{th v-type layer}\}.$$

When $M(D,e)=\emptyset$: By the definition of level types, the top layer in D contains at least two levels unless D is composed of one x-type level only.

When D is composed of only one x-type level with label x, H must be $[(\emptyset,\emptyset,a)]$, where $a=0$ or 1. In this case, the layer list of D is $[(\emptyset,\{x\},\bar{a}\oplus x(e))]$ because $D(e)=\bar{a}$. In the procedure, $S=\{x\}$ at Line 04, $|S|=1$ at Line 5, and $W_x=\emptyset$ at Line 07. Since $l=0$ at Line 09, the procedure outputs the layer list of D at Line 10.

Assume that the top layer in D contains at least two levels. In order to avoid dealing exceptionally with the case that $H=[(\emptyset,\emptyset,0)]$ or $[(\emptyset,\emptyset,1)]$, we let $l'=-1$ when $H=[(\emptyset,\emptyset,0)]$ or $[(\emptyset,\emptyset,1)]$ and let $l'=l$ otherwise. We define $B(D,e,i_L,i_U)$ as follows:

$$B(D,e,i_L,i_U) \triangleq \{i\text{: } i_L<i<i_U,\ \text{Path}(D,e) \text{ contains branching edges in the } i\text{th v-type layer}\}.$$

At Line 04 $S=\bigcup_{i>l'} X_i^* \cup \bigcup_{i\in B(D,e,l',\infty)} V_i^{*\prime}$ trivially holds. Since S includes all the variables in the top layer, $|S|\geqslant 2$ at Line 05. Thus, Lines 06–12 are not executed. For each $x\in S$, the following holds at Line 13:

$$S_x \triangleq \{y\in S-\{x\}: D(e_{\overline{x},\overline{y}})\neq D(e_{\overline{x}})\} = \begin{cases} \left(\bigcup_{j>l'} X_j^* - \{x\}\right) \cup \bigcup_{i<j\in B(D,e,l',\infty)} V_i^{*\prime} & \text{if } x\in X_i^*, \\ \bigcup_{l'<j<i} X_j^* & \text{if } x\in V_i^{*\prime}. \end{cases} \tag{1}$$

Let us prove

$$V \triangleq \{x\in S: \exists y\in S \text{ s.t. } x\neq y, S_x\subseteq S_y\} = \bigcup_{i\in B(D,e,l',\infty)} V_i^{*\prime} \tag{2}$$

at Line 14.

Assume that $x\in\bigcup_{i>l'} X_i^*$. Then, for all $y\in\bigcup_{i>l'} X_i^*$, $S_x\not\subseteq S_y$ unless $x=y$, because $y\in S_x$ and $y\notin S_y$. Let $y\in\bigcup_{i\in B(D,e,l',\infty)} V_i^{*\prime}$. If $X_d^*\neq\emptyset$, $|X_d^*|\geqslant 2$. Then $S_x\not\subseteq S_y$ because $\emptyset\neq X_d^*-\{x\}\subseteq S_x$ and $X_d^*-\{x\}\not\subseteq S_y$. If $X_d^*=\emptyset$, $d\in B(D,e,l',\infty)$. Then $S_x\not\subseteq S_y$ because $V_d^{*\prime}\subseteq S_x$ and $V_d^{*\prime}\not\subseteq S_y$. Thus $x\notin V$.

Assume that $x\in\bigcup_{i\in B(D,e,l',\infty)} V_i^{*\prime}$. Let $x\in V_i^{*\prime}$. If $|\bigcup_{i\leqslant j\in B(D,e,l',\infty)} V_j^{*\prime}|\geqslant 2$, then $S_x\subseteq S_y$ for $y\in\bigcup_{i\leqslant j\in B(D,e,l',\infty)} V_j^{*\prime}-\{x\}$. If $|\bigcup_{i\leqslant j\in B(D,e,l',\infty)} V_j^{*\prime}|=1$, then $X_d^*\neq\emptyset$ because, if $X_d^*=\emptyset$, then $d\in B(D,e,l',\infty)$ and $|V_d^{*\prime}|\geqslant 2$, which would be a contradiction. Thus, in this case, $S_x\subseteq S_y$ for $y\in X_c^*$. Therefore $x\in V$. This concludes the proof of (2).

Let us define (i_g,j_g) inductively as follows:

$$i_1=\min B(D,e,l',\infty), \quad i_g=\min\{j\in B(D,e,l',\infty): j_{g-1}<j\},$$

$$j_g=\max(\{j\in B(D,e,l',\infty): X_f^*=\emptyset \text{ for } i_g\leqslant f<j\}\cup\{i_g\}).$$

Let (i_{k^*},j_{k^*}) be the last well-defined pair. By (1) and (2), for $(k,(S_1,V_1),\ldots,(S_k,V_k))$ constructed at Line 15, we obtain $k=k^*$ and

$$(S_g,V_g)=\left(\bigcup_{l'<j<i_g} X_j^*, \bigcup_{j\in B(D,e,l',\infty), i_g\leqslant j\leqslant j_g} V_j^{*\prime}\right)$$

for all $g\in\{1,\ldots,k\}$. Let $j_0=l'+1$ for convenience. Then,

$$(R_{2g-1},t_{2g-1})=\left(\bigcup_{j_{g-1}\leqslant j<i_g} X_j^*, \text{`x'}\right),\ (R_{2g},t_{2g})=\left(\bigcup_{j\in B(D,e,l',\infty), i_g\leqslant j\leqslant j_g} V_j^{*\prime}, \text{`b'}\right)$$

for all $g\in\{1,\ldots,k\}$ at Line 17 and $(R_{2k+1},t_{2k+1})=(\bigcup_{j_{k^*}\leqslant j} X_j^*, \text{`x'})$ at Line 18. Therefore, for each $x\in R_{2g-1}$,

$$W_x \triangleq \{y\in Z: D(e_{\overline{x},\overline{y}})\neq D(e_{\overline{x}})\} = \bigcup_{j\notin B(D,e,l',\infty), j_{g-1}\leqslant j<i} V_j^{*\prime} \quad \text{for } x\in X_i^*,$$

and for each $x \in R_{2g}$,

$$W_x = \bigcup_{j \notin B(D,e,l',\infty), i_g \leqslant j < i} V_j^{*\prime} \quad \text{for } x \in V_i^{*\prime}$$

at Line 23. Thus, it is trivial that all non-empty layers which are not in H are listed in L at Line 25 with their order of appearance being that in D as taken from bottom up.

A layer list of D is recovered correctly from H and L for the following reasons. Assume that (V_i^*, X_i^*, a_i^*) is recovered correctly for all $i < j$. Then, we prove that (V_j^*, X_j^*, a_j^*) is also recovered correctly. Note that the following proof is applicable even when $j = 0$. Let $x \in V_j^{*\prime}$. Since Path(D,e) contains a non-merging edge outgoing from the x-level, the label of merging edge of this level is $\overline{x(e)}$. Thus $x^{\overline{x(e)}} \in V_j^*$. Therefore V_j^* is recovered correctly at Line 32. It is trivial that X_j^* is recovered correctly at Line 31. Note that a is so updated at Line 31 that Path(D,e) contains a node on the a-lane at the top level of the topmost of those layers which have already been recovered. If $j+1 \in B(D,e,l',\infty)$, then the $(j+1)$th v-type layer is $\overline{a}$-v-type, therefore $a_j = \overline{a}$. If $j+1 \notin B(D,e,l',\infty)$, then that layer is a-v-type, therefore $a_j = a$. Thus, a_j is also recovered correctly at Line 33.

Thus, H defined at Line 35 is the layer list of D. Since Y_m is the set of variables that appear in the top layer, $|Y_m| \geqslant 2$. Therefore, the procedure outputs the layer list of D at Line 39.

When $M(D,e) \neq \emptyset$: Let $i' = \min M(D,e)$. Let x' be a variable labelling the node in the i'th v-type layer from which the (branching) merging edge contained in Path(D,e) goes out. Then $S = \bigcup_{i' > i > l'} X_i^* \cup \bigcup_{i \in B(D,e,l',i')} V_i^{*\prime} \cup \{x'\}$ at Line 04.

When $|S| \geqslant 2$:

In this case, $i' > l+1$. Let $D_{x'}$ be the normal-form width-2 OBDD composed of all the layers below the x'-level in D and the x'-level itself, namely, the width-2 OBDD represented by the layer list $[(\emptyset, X_0^*, a_0^*), \ldots, (V_{i'-1}^*, X_{i'-1}^* \cup \{x'\}, a_{i'-1}^* \oplus x'(e))]$ when $X_{i'-1}^* \neq \emptyset$ and by $[(\emptyset, X_0^*, a_0^*), \ldots, (V_{i'-1}^* \cup \{x'^{\overline{x'(e)}}\}, \emptyset, a_{i'-1}^*)]$ otherwise. There are three lines at which membership queries are asked in this case: Line 04 to construct S, Line 13 to construct S_x, and Line 23 to construct W_x. The differences between the answers to the membership queries for D and $D_{x'}$ appear only at Line 23 for $W_{x'}$. For D,

$$W_{x'} = W_z \cup \left(\bigcup_{i'' > i > i'-1} X_i^* \right) \cup \left(\bigcup_{i \in B(D, e_{\overline{x'}}, i'-1, i'')} V_i^{*\prime} \right) \cup Z - \{x'\},$$

where

$$z \in \begin{cases} X_{i'-1}^* & \text{if } X_{i'-1}^* \neq \emptyset, \\ V_{i'-1}^{*\prime} & \text{otherwise,} \end{cases}$$

$$i'' = \min(M(D, e_{\overline{x'}}) \cup \{\infty\}) = \min(M(D,e) \cup \{\infty\} - \{i'\}), \text{ and}$$

$$Z = \begin{cases} \{x'' : x'' \text{ is a variable labelling the node in the } i''\text{th} \\ \text{v-type layer from which the (branching) merging} & \text{if } i'' \neq \infty, \\ \text{edge contained in Path}(D,e) \text{ goes out} \\ \emptyset & \text{otherwise.} \end{cases}$$

We get $W_{x'} - W_z \neq \emptyset$ because $\emptyset \neq Z \subseteq W_{x'}$ if $i'' \neq \infty$ and the variables in the top layer are contained in $W_{x'}$ otherwise. Let L at Line 27 for $D_{x'}$ be $[(Y'_1, y'_1), \ldots, (Y'_{m'}, y'_{m'})]$. Then, L at Line 27 for D is $[(Y'_1, y'_1), \ldots, (Y'_{m'-1}, y'_{m'-1}), (Y'_{m'} - \{x'\}, y'_{m'})(W_{x'} - W_z, \text{'n'}), (\{x'\}, \text{'b'})]$. Thus, $H = [(, X_0^*, a_0^*), \ldots, (V_{i'-1}^*, X_{i'-1}^*, a_{i'-1}^*)]$ at Line 35. Let H' be this H. Then, Construct-normalized-sublist$(H', e_{\overline{x'}})$ is executed at Line 38. Since $|M(D, e_{\overline{x'}})| = |M(D,e)| - 1$ and $|S(H', e_{\overline{x'}})| \geqslant 2$ at Line 04, after $|M(D,e)|$ recursive calls of the procedure, $M(D,\cdot)$ will become $\emptyset$ and we can see, from the proof for the case with $M(D,e) = \emptyset$, that the algorithm will output the layer list of D.

When $|S| = 1$: In this case, $i' = l + 1$, and S must be $\{x'\}$. Thus, $W_{x'} \neq \emptyset$ at Line 07 for the same reason as $W_{x'} - W_z \neq \emptyset$ proved above. $a_l \neq a_{i'-1}^*$ must hold, which means that H is not a layer sublist of D, namely, $H = [(\emptyset, \emptyset, 0)]$. Let H' be a list constructed from H by only flipping the bit of a_l at Line 12, that is to say, $H' = [(\emptyset, \emptyset, 1)]$. Then, H' is a layer sublist of D. Since $S(H', e_{\overline{x'}}) = \{x'\} \cup W_{x'}$, we get $|S(H', e_{\overline{x'}})| \geqslant 2$. Thus, this case is reduced to the case with $|S| \geqslant 2$. □

Theorem 11. *Procedure W2-OBDD-QLearning learns width-2 OBDDs in time polynomial in n using* $O(n)$ *proper equivalence queries and* $O(n^2)$ *membership queries.*

Proof. By Lemmas 8–10, the procedure outputs the layer list of D.

The number of equivalence queries and polynomial time complexity are trivial. Let us consider the number of membership queries. In procedure W2-OBDD-QLearning, at most n membership queries are used to construct $S(H,e)$. Since the repetition in this procedure is at most n times, $O(n^2)$ membership queries are asked here. In procedure Denormalization, there are four lines at which membership queries are asked: Lines 04, 07, 13 and 15. Since all the lines need at most n membership queries and this procedure is called at most n times, $O(n^2)$ membership queries are used. In procedure Construct-normalized-sublist, membership queries are used at Lines 04, 07, 13 and 23. Since this procedure is also called at most n times, we only have to check the number of queries used at Lines 13 and 23 to prove that the total number of queries is $O(n^2)$. Note that all the variables in S constructed at Line 04 except at most one variable are used to construct the updated H. Thus, S contains at most one old member variable in each call. Therefore, for each variable pair (x,y), the values at $e_{\overline{x},\overline{y}}$ are never asked more than once at Line 13 through the execution of procedure W2-OBDD-QLearning. This means that the number of queries asked at this line is $O(n^2)$. At Line 23, the number of queries needed to construct W_x for all new member variables x is trivially $O(n^2)$ in total. For an old member variable, the procedure needs at most n membership queries. Thus, $O(n^2)$ queries are used in total for old member variables at Line 23. □

6. Hardness result on learning width-3 OBDDs via proper equivalence queries alone

Angluin et al. [2] proved that the class of read-once formulas[7] is not learnable in polynomial time via proper equivalence queries alone. Raghavan and Wilkins [9] used the same method to show that the same is true for the class of μ-branching programs. Here, we used the method to show that the same is true for the class of width-3 OBDDs.

In order to prove such a hardness result, we show that there exists an adversary strategy of providing counterexamples which allows no algorithm to learn using polynomial number of equivalence queries. The next lemma guarantees that, for any hypothesis OBDD, there is an counterexample which assigns 0 to a small number of variables or assigns 1 to a small number of variables.

Lemma 12 (Lemma 1 in Raghavan and Wilkins [9]). *For an OBDD composed of N nodes, let r_0 denote the minimum number of* 0*-labelled edges contained in any one path among all paths from the root node to the* 0*-labelled sink node, and let r_1 denote the minimum number of* 1*-labelled edges contained in any one path among all paths from the root node to the* 1*-labelled sink node. Then, $N \geqslant r_0 \cdot r_1$.*

Assume that the number of variables n is the square of a natural number m. Then, by Lemma 12, any width-3 OBDD has either a path from the root node to the 0-labelled sink node with at most m 0-labelled edges or a path from the root node to the 1-labelled sink node with at most $3m$ 1-labelled edges.

Let H_n be the class of monotone read-once DNF formulas composed of just m terms, each of which has just m variables. The class of functions represented by formulas in H_n is included in the class of functions represented by width-3 OBDDs[8] as well as in the class of functions represented by read-once formulas and in the class of functions represented by μ-branching programs. We use H_n as the class from which a target function is assumed to be selected in our proof, as Angluin et al. and Raghavan et al. have done. For each permutation of $n = m^2$ variables, let H_n have one formula regarding the ith group of m variables as the ith term. Note that H_n has many logically equivalent formulas.

For an assignment a with at most $3m$ 1s, the number of formulas f in H_n satisfying $f(a) = 1$ is at most

$$\binom{3m}{m} mm!(n-m)!$$

and for an assignment a with at most m 0s, the number of formulas f in H_n satisfying $f(a) = 0$ is at most $m^m m!(n-m)!$. If m is large enough, the former is also at most

[7] Read-once formulas are Boolean formulas over the basis $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT) that contain at most one occurrence of each variable.

[8] Note that the class of functions represented by width-2 OBDDs does not include the class of functions represented by formulas in H_n, and that fact prevents the application of the following argument to that class.

$m^m m!(n-m)!$. Thus, given a counterexample with at most $3m$ 1s or at most m 0s, it reduces the number of consistent formulas in H_n by at most $m^m m!(n-m)!$. If the adversary gives such counterexamples only, then the number of equivalence queries needed to learn is not polynomial in n, because the number of formulas in H_n is $n!$, the number of logically equivalent formulas in H_n is $(m!)^{m+1}$, and the following lemma holds.

Lemma 13 (Lemma 18 in Angluin et al. [2]). *For any constant $C>1$ and for any sufficiently large integer m, if $n=m^2$ then*

$$\frac{m^m m!(n-m)!}{n!} \leqslant C\sqrt{2\pi(m-1)}\mathrm{e}^{-(m-1)}.$$

Thus, Theorem 15 is obtainable by the same argument as Theorem 14.

Theorem 14 (Theorem 19 in Angluin et al. [2]). *There is no polynomial-time algorithm that exactly identifies all read-once formulas using equivalence queries. In fact, there is no polynomial-time algorithm that exactly identifies the class of monotone read-once formulas in disjunctive normal form, even if the equivalence queries are allowed to consist of arbitrary read-once formulas.*

Theorem 15. *There exists no polynomial-time algorithm that learns width-3 OBDDs via proper equivalence queries. Furthermore, there exists no polynomial-time algorithm that learns monotone read-once DNFs via equivalence queries using width-3 OBDDs as hypotheses.*

Proof. The only difference from the proof of Theorem 19 in [2] is the strategy taken by the adversary to choose its counterexamples. Here, we need only explain the strategy.

Consider the following adversary strategy. To an equivalence query with a hypothesis width-3 OBDD H, it answers 'NO' and returns a counterexample in the following way. Let a_0 be an assignment with the smallest number of 0's satisfying $H(a_0)=0$, and let a_1 be an assignment with the smallest number of 1's satisfying $H(a_1)=1$. The adversary returns a_0 if the number of 0's in a_0 is at most m, and returns a_1 if the number of 1's in a_1 is at most $3m$. At least one of these two conditions is satisfied by Lemma 12. □

7. Concluding remarks and open questions

In this paper, we have studied query learnability of bounded-width OBDDs in the case with an unknown variable ordering. We have shown that the class of width-2 OBDDs is learnable from equivalence queries alone and the class of width-3 OBDDs is not learnable from proper equivalence queries alone. These results lead one to the following open question: are there hypothesis classes which make the class of width-3 OBDDs learnable? Note that OBDDs are *read-once* branching programs. Here,

read-once branching programs differ from μ-branching programs. Read-once restriction on branching programs is that any variable appears at most once as labels of nodes on any path from the root to one of the sink node. If this read-once restriction is removed, any hypothesis class seems not to be able to make the class of width-3 branching programs learnable, even by using membership queries, because learning DNFs [7] is reducible to learning width-3 branching programs, and learning DNFs is believed to be difficult and cannot be helped by membership queries [3]. As a more general question, is the class of width-k OBDDs learnable for any fixed k?

Acknowledgements

We wish to thank Dr. Naoki Abe of C&C Research Laboratories for his helpful comments and advice. We are also grateful to Mr. C. Tamon of the University of Calgary for sharing so much information from his research on learning branching programs. We would also like to thank the anonymous referees for their valuable comments to improve this paper.

Appendix A. Effect of variable ordering

Let f be a Boolean function on $X=\{(x_1,\ldots,x_n)\colon\ x_1,\ldots,x_n\in\{0,1\}\}$. For variable ordering π on $\{x_1,\ldots,x_n\}$, the *π-OBDD-width of f* is the minimum width among widths of all levelled OBDDs with ordering π which represent f. From the reduced OBDD D^r with ordering π which represents f, we can construct a levelled OBDD D^l having the minimum width by procedure Transform-to-levelled-OBDD. Let $\{x_{i_1},\ldots,x_{i_k}\}$ be the set of variables labelling the nodes in D^r and assume $x_{i_1}<\cdots<x_{i_k}$ in ordering π. For convenience sake, we call the set of sink nodes the $x_{i_{k+1}}$-level.

Transform-to-levelled-OBDD: Let $D=D^r$ initially. Repeat the procedure *Replace-one-edge* until there are no outgoing edges in D from the x_{i_j}-level into the x_{i_l}-level with $l>j+1$.

Replace-one-edge: Let E be an outgoing edge from node N_j in the x_{i_j}-level into N_l in the x_{i_l}-level with $l>j+1$.

(1) Create new nodes N_h labelled x_{i_h} for $h=j+1,\ldots,l-1$.
(2) Redirect E to N_{j+1}.
(3) Make an a-labelled outgoing edge from N_h into N_{h+1} for $h=j+1,\ldots,l-1$ and $a=0,1$.
(4) For all $m\in\{1,\ldots,l-1\}$, redirect all outgoing edges from x_{i_m}-level into N_l to N_h, where $h=\min\{d\in\{j+1,\ldots,l\}\colon\ d>m\}$.

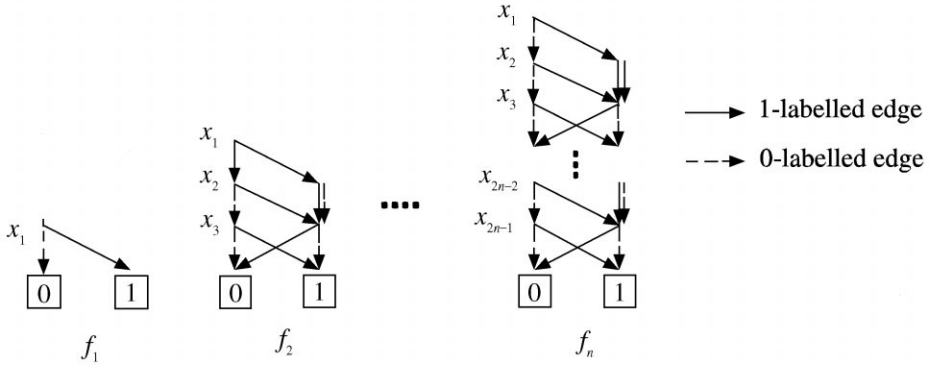
Let us define $W_D^{\pi}(y)$ as the number of nodes N in D satisfying that $y\leqslant z$ holds for N's label z and there exists a variable x such that $x<y$ and at least one of the edges outgoing from the x-level enters N. Assume that D changes into D' by applying procedure Replace-one-edge once. Procedure Replace-one-edge has the following properties.

Property 1. D' represents the same function as D.

Property 2. $W^{\pi}_{D'}(x) = W^{\pi}_{D}(x)$ for all variables x.

Let D^r be a reduced OBDD with ordering π which represents f, and let D^l be the levelled OBDD created from D^r by procedure Transform-to-levelled-OBDD. By Property 1, D^l represents f. Let us show that the width of D^l is really the minimum among widths of all levelled OBDDs representing f with ordering π. Assume that there is a levelled OBDD D representing f with ordering π whose width is smaller than D^l. Let the x-level be one of the levels containing the maximum number of nodes in D^l. Then, $W^{\pi}_{D}(x) < W^{\pi}_{D^l}(x) = W^{\pi}_{D^r}(x)$. By applying the reduction rules in [5], D is transformed into D^r. Thus, $W^{\pi}_{D^r}(x) \leqslant W^{\pi}_{D}(x)$ trivially. This leads to the contradiction that $W^{\pi}_{D}(x) < W^{\pi}_{D}(x)$. Therefore, there are no OBDDs representing f with ordering π whose width is smaller than D^l. Hence, the π-OBDD-width of f is $\max_x W^{\pi}_{D^r}(x)$.

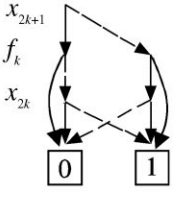
Here, we show the existence of a function sequence $f_1, f_2, \ldots$ and a variable ordering sequence $\pi_1, \pi'_1, \pi_2, \pi'_2, \ldots$ such that the π_n-OBDD-width of f_n is two, but its π'_n-OBDD-width is exponential in the number of variables of f_n. Define a function sequence $f_1, \ldots, f_n, \ldots$ representable by width-2 OBDDs with variable ordering π_n $(x_1 < x_2 < \cdots < x_{2n-1})$ as follows:



Theorem 16. *The π'_n-OBDD-width of f_n is 2^n for variable ordering π'_n defined as* $x_{2n-1} < x_{2n-3} < \cdots < x_3 < x_1 < x_2 < x_4 < \cdots < x_{2n-2}$.

Proof. It is trivial when $n = 1$. Assume that the theorem statement holds when $n = k$.

Function f_{k+1} can be represented by the following:



Consider OBDD D_{k+1} with ordering π'_{k+1} constructed from this graph by replacing f_k with the reduced OBDD D_k with ordering π'_k which represents f_k. Let x be a variable satisfying $W_{D_k}^{\pi'_k}(x) = \max_y W_{D_k}^{\pi'_k}(y)$. Then, $W_{D_k}^{\pi'_k}(x) = 2^k$ by the inductive hypothesis. Since D_{k+1} is also the reduced OBDD, $W_{D_{k+1}}^{\pi'_{k+1}}(x_i) \leqslant W_{D_{k+1}}^{\pi'_{k+1}}(x) = 2^{k+1}$ for $i = 1, \ldots, 2k-1$, $W_{D_{k+1}}^{\pi'_{k+1}}(x_{2k}) = 4$ and $W_{D_{k+1}}^{\pi'_{k+1}}(x_{2k+1}) = 1$. Thus, the π'_{k+1}-OBDD-width of f_{k+1} is 2^{k+1}. □

References

[1] D. Angluin, Learning regular sets from queries and counterexamples, Inform. Comput. 75 (1987) 87–106.
[2] D. Angluin, L. Hellerstein, M. Karpinski, Learning read-once formulas with queries, J. ACM 40 (1) (1993) 185–210.
[3] D. Angluin, M. Kharitonov, When won't Membership queries help?, J. Comput. System Sci. 50 (1995) 336–355.
[4] F. Bergadano, N.H. Bshouty, C. Tamon, S. Varricchio, On learning branching programs and small circuits, unpublished manuscript.
[5] R.E. Bryant, Symbolic boolean manipulation with ordered binary decision diagrams, ACM Comput. Surveys 24 (1992) 293–318.
[6] N.H. Bshouty, C. Tamon, D.K. Wilson, On learning width two branching programs, in: Proc. of 9th Ann. Conf. on Comput. Learning Theory, ACM, New York, 1996, pp. 224–227.
[7] F. Ergün, S. Kumar, R. Rubinfeld, On learning bounded-width branching programs, in: Proc. of 8th Ann. Conf. on Comput. Learning Theory, ACM, New York, 1995, pp. 361–368.
[8] R. Gavaldà, D. Guijarro, Learning ordered binary decision diagrams, in: Proc. of 6th Internat. Workshop on Algorithmic Learning Theory. Lecture Notes in Artificial Intelligence, vol. 997, Springer, Berlin, 1995, pp. 228–238.
[9] V. Raghavan, D. Wilkins, Learning μ-branching programs with queries, in: Proc. of 6th Ann. Conf. on Comput. Learning Theory, ACM, New York, 1993, pp. 27–36.
[10] H.U. Simon, Learning decision lists and trees with equivalence-queries, in: Proc. of 2nd European Conf. on Comput. Learning Theory, Lecture Notes in Artificial Intelligence, vol. 904, Springer, Berlin, 1995, 322–336.